Canada $4.25
USA $3.50

# FOR THE Amiga Transactor

OCTOBER 1989: VOLUME 3, ISSUE 1

THE MAGAZINE FOR AMIGA PROGRAMMERS

**Introduction to Usenet** ***by Tim Grantham***

**Technicolour text** ***by Anthony Bryant***

**Disk encoding schemes** ***by Betty Clay***

**PLUS:**

- Writing a compiler - the code generator *by Chris Gray*
- Avoiding memory fragmentation *by Nick Sullivan*
- The Amiga versus other PCs *by Ben Williams*
- Automate operations with *Scriptit* *by Steve Ahlstrom*
- Regular columns by Jim Butterfield, Don Curtis and Derek Grime

**Reviews:** *IMG Scan* scanner, *MediaLine Animation Backgrounds and Clip Art*

*Fashion* by Gregg Bailey

0 29074 12122 10

## Lattice Tools & Libraries

We are the company that writes the language and the tools. So who could know more about the utilities you need to maximize productivity?

**Lattice 5.0 Cross Compiler**
All the power and facilities of the Lattice AmigaDOS 5.0 compiler in a cross-compiler MS-DOS and OS/2 host. Includes our full screen cross debugger. $750.

**Lattice C++**
Object Oriented C++ Programming for the Amiga. Provides object definitions for AmigaDOS, EXEC and Intuition. $300.

**Lattice Compiler Companion**
Collection of UNIX-like utilities that make your programming environment more productive. (Included in Lattice 5.0.) $100.

**dBC III Library**
Amiga programming library that provides programming interface into dBASE III compatible files. $150.

**Communication Library**
New Amiga programming library that supports modem interfaces for XMODEM, YMODEM, KERMIT and ASCII protocols. $250.

**PANEL Library**
Amiga programming library that supports windows graphics applications. $195.

**C Programming Seminar**
Learn C programming from Lattice experts. Four day seminar in Chicago. $895.

**Call any of our world-wide distributors, or call us at 1-800-444-4309 or (312) 916-1600. Fax #(312) 916-1190, TELEX 532253.**

# "Brilliant!"

– *William Hawes* author of ARexx

# "Lightning Fast!"

– *Reichart Von Wolfsheild* Silent Software, Inc.

# "Blows away all the others."

– *Robert Berryhill* LMR Creations

**Lattice AmigaDOS C 5.0** is getting lots of praise. And not just because of the latest benchmarks which prove Lattice C 5.0 is the best compiler by every measure.

For instance, Michael Berenstein calls 5.0 "Wonderful," while Roger Uzun simply calls this total programming environment "...the best."

Tom Dolan felt the same about CodePRobe, our new Source Level Debugger. Support for both C and Assembly language, plus multi-tasking debugging make it "...the best I have seen or used!!!!"

And our new Global Optimizer caused Steve Tibbett, a "VERY happy customer," to say, "WOW. This is outstanding!" and Tomas Rokicki of Radical Eye Software to exclaim "...Global Optimizer, inline subroutines and extensive libraries make whole new levels of performance possible."

About our manual? "Excellent!" And "...extra fine."

About our BBS for unlimited technical support? "Fantastic!"

And about our competition? Eric Dyke of Inom Software says, "Manx, eat your heart out! Lattice is the right choice!"

**Lattice**

**Professional Programming Tools Since 1981**
2500 S. Highland Ave., Lombard, IL 60148

# TOM & JERRY

## Hunting High and Low

AMIGA

AMIGA

AMIGA

**Play a little game of cat and mouse with Tom & Jerry!**

**It'll drive you wild!**

DISTRIBUTED BY
**INNERPRISE**
SOFTWARE, INC
128 COCKEYSVILLE ROAD
HUNT VALLEY, MD 21030
(301) 785-2266

**Available for: AMIGA, C-64 & Atari ST**

©1989 Turner Entertainment, Inc.
all rights reserved

MAGIC BYTES

Volume 3, Issue 1

**Publisher**
Antony Jacobson

**Vice-President Operations**
Jeannie Lawrence

**Assistant Advertising Manager**
Mike Grantham

**Editors**
Chris Zamara
Nick Sullivan
Tim Grantham

**Assistant Editors**
Malcolm O'Brien
Paul Bosacki

**Contributing Writers**
Steve Ahlstrom
Scott Ballantyne
Richie Bielak
Daniel Brookshier
Anthony Bryant
Jim Butterfield
Betty Clay
Don Curtis
Matt Dillon
Andy Finkel
Brad Fowles
Eric Giguère
Cathryn Graham
Eric Graham
Chris Gray
Derek Grime
Jay Gross
Eric Haberfellner
Mike Hubbartt
James C. Lawrie
Hobie Orris
Rob Peck
Larry Phillips
Bob Rakosky
Dan Schein
Mike Schwartz
Steve Simpson
John Toebes
Victor A. Wagner
Doug Walker
Ben Wiliams
David Wood

**Cover Artist**
Gregg Bailey

for the AMIGA Transactor
The Magazine for Amiga Programmers

# Features

## Programming In M2AMIGA — 34
by Anthony Bryant
Dazzling technicolour text effects.

## Toy: A Small Compiler For The Amiga — 40
by Chris Gray
In the fourth part of the Toy series, Chris dissects the compiler's assembly language code generator.

## Introduction to Usenet — 50
by Tim Grantham
A guide for the novice to this global network, and a compendium of ample answers to common questions about UNIX, the Amiga and life after death.

## Allocate and Deallocate — 62
by Nick Sullivan
Nick shows how to get around memory fragmentation problems using EXEC's own fundamental memory management functions.

## Pass The Ammo! — 67
by Ben Williams
"My computer is better than your computer, so there!" If that's all you have to say, then Ben's got the ammunition you need to win the argument next time.

## Amiga Disk Encoding Schemes — 70
by Betty Clay
We welcome Betty back to these pages with her article on the MFM and GCR floppy disk data encoding schemes. Did you know that the Amiga can read and write both formats?

# Departments and Columns


## AbsExecBase 6

A salute to the man behind the late, lamented *The Amigan Apprentice and Journeyman*.

## Amiga Dispatches 13

by Don Curtis

Don ponders the future of 'home computers' and welcomes exciting developments in Commodore's marketing strategy.

## ACCESS 17

by Steve Ahlstrom

You can automate your interaction with applications with Khalid Aldoseri's new scripting language *Scripit*. Steve shows you how to use this freely distributable program.

## Mastering The Raster 20

by Derek Grime

Derek rebutts himself with this month's appreciation of the Amiga's commanding lead in the desktop video market.

## Assembly Language Programming 24

by Jim Butterfield

Part two of Jim's examination of object files, linking, sections and program units.

## NewsLINK 79


# Reviews


**MediaLine CLipArt** from Free Spirit Software 61

**MediaLine Animation Backgrounds** from Free Spirit Software 32

**IMG Scan** from SunRize Industries 74


**About the cover:** *Fashion* by Gregg Bailey

Gregg is a 25-year old art student at Indiana University in Fort Wayne, IN. The original idea for *Fashion* was taken from a local restaurant advertisement. Gregg first drew it on paper, added the rose and then digitized the picture. Pixmate was used to change the colour scheme and the image was finished in Photon Paint. Gregg will receive his degree in commercial art in the fall.


TransAmi's phone number is:
(416) 764-5273

FAX: (416) 764-9262

**TOLL-FREE ORDER LINE**
**1-800-248-2719 Extension 911**
(orders only; have your VISA or Mastercard number ready; available in the U.S. only)

Transactor for the Amiga is published monthly by Croftward Publishing Inc., 10-85 West Wilmot Street, Richmond Hill, Ontario, L4B 1K7. ISSN# 0838-0163. Canadian Second Class Mail Registration No. 7690, Gateway-Mississauga, Ont. USPS Postmaster: send address changes to: Transactor, PO Box 338, Station C, Buffalo, NY, 14209.



Croftward Publishing Inc. is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names are registered trademarks of Commodore Inc.



**Subscriptions:** see order card at centre of magazine for complete ordering information and prices.

**Send all subscriptions to:** TransAmi Subscriptions, 10-85 West Wilmot Street, Richmond Hill, Ontario, Canada, L4B 1K7. For best results, use the postage paid card at the centre of the magazine.

**Quantity Orders:** In Canada: Disticor, 155 Gordon Baker Road Suite 213, North York, Ontario M2H 3N7. Tel (416) 496-8444 Fax (416) 496-8448. In the USA: IPD (International Periodical Distributors), 11760-B Sorrento Valley Road, San Diego, California, 92121, (619) 481-5928; ask for Dave Buescher.

**Editorial contributions**: writer's guides are available by writing to the Richmond Hill address above. Write to the editorial department (same address) with article proposals or articles for consideration.



All material accepted becomes the property of Croftward Publishing Inc., except by special arrangement. All material is copyright by Croftward Publishing Inc. Reproduction in any form without permission is in violation of applicable laws. Write to the Richmond Hill address for a writer's guide.

The opinions expressed in contributed articles are not necessarily those of Croftward Publishing Inc. Although accuracy is a major objective, Croftward Publishing Inc. cannot assume liability for errors in articles or programs. Programs listed in Transactor for the Amiga, and/or appearing on Transactor for the Amiga disks, are copyright by Croftward Publishing Inc. and may not be duplicated or distributed without permission.



**Transactor for the Amiga** is produced using Amigas and the following products:

Professional Page
Deluxe Paint II
Transcript
Digi-View

Final output by Vellum Print & Graphic Services, Inc., Toronto

Printed in Canada by Bowne of Toronto, Inc.

# What is CanDo?

CanDo is interactive audio-visual authoring software that lets anyone create: Prototyped Software, Desktop Video, Multimedia Presentations, Interactive Presentations, Tutorials, Educational applications, Utilities, Recreational software, vertical market applications, and a whole lot more!

- CanDo is a significant step forward in Amiga software.
- It is an application generator which integrates the fundamental elements of the Amiga.
- It gives Amiga users a practical way to expand the output of popular software packages.
- You interactively combine the IFF files generated by a host of popular software packages to create real Amiga software with menus, windows, gadgets, and timers.
- CanDo's powerful scripting language allows you to easily perform complex tasks that usually require extensive development time using traditional languages.
- Professional developers can use CanDo to prototype software packages.
- Your programs can be saved in a re-editable format or as a standalone program.
- Programming was never like this!

**CanDo utilizes files from the following Software packages:**

- **Graphics:** Deluxe Paint III, Photon Paint, DigiPaint, Images and all other ILBM generating paint programs.
- **Video:** DigiView, Calligrapher, Perfect Vision, Zuma Fonts, ImageScan, FontWorks, and many more.
- **Sound:** AudioMaster, PerfectSound, FutureSound, etc.
- **CanDo can directly control the following software packages:** ARexx, ATalk 3, SuperBase, SuperPlan, CygnusEd, C.A.P.E.68k, Benchmark Modula-2, and any other software which supports the ARexx language. (ARexx is not required.)
- **IFF forms supported include:** ILBM, 8SVX, GRAB and BrushANIMs
- **Amiga Hardware Requirements:** Any Amiga with 1 MB RAM (A500, A1000, A2000, A2500)
- **Other Hardware supported:** Any Genlock, Magni 4004, SuperGen and with custom micro servers you can control, LaserDisc players, VCRs, Video Editing decks, and more.

**CanDo**

Power Steering For The Amiga

**Anyone that uses an Amiga needs CanDo!**

**Retail Price: $149.95**

CanDo is a product of Inovatronics, Inc. 8499 Greenville Avenue, Suite 209B Dallas, Texas 75231 214-340-4991
Dealers and Distributors should contact: Southern Technologies, 800-647-7741, in Texas 214-247-7373

CanDo and Power Steering for the Amiga ©1989 Inovatronics Inc. Amiga is a registered trademark of Commodore-Amiga Inc.
DELUXE PAINT III is a trademark of Electronic Arts Inc.; DIGI-VIEW is a trademark of Newtek Inc.;
AUDIO MASTER II is a trademark of Aegis Development Inc.; ARexx is copyrighted by William S. Hawes.

# It's new. It's hot. It's here.

# Imtronics HD -Series.

- **Autobooting from FFS-Partition under Kickstart 1.3**
- **No DMA trouble!**
- **No formatting horror trip, comes ready to use!**
- **Up to 8 megabyte RAM, user-expandable!**
(HD4000/ 8000 only)

**33 Mb Harddrive for A500 as low as $698.00**

**33 Mb Hardcard for A2000 as low as $649.00**

**1 Mb RAM only $198.00**

## Don't wait any longer - get the power you can afford today.

The Imtronics HD-Series offers you a complete product-line of full autobooting, ultrafast 3,5" Hard-drives which are available for both the Amiga 500 and - as Hardcards - for the Amiga 2000. These drives provide a high-speed non-DMA design which prevents you from any trouble DMA-devices may cause, as well as optional memory expansion up to 8 Mb on board. Other features include passed-through expansion bus and compact Amiga-shaped case (models for A500 only), different capacities from 33 up to 84 megabytes, the possibility to control additional external drives and full autobooting capabilitiy from a FFS-partition.

If you are looking for a hard drive system that offers you superior power for a low budget, the Imtronics HD Series will be your first choice.

Ask your favorite dealer today, or call us for further information.

**Imtronics, Inc.**
12301 South West 132 Court
Miami, Florida 33186
Phone 305 255 9302 or 6903
FAX 305 255 6903

Distributed by American Software
Call Toll Free 800 225 7941

# AbsExecBase

## *A tip o' the hat to Dick Barnes*

Though he would likely be very embarassed to read this, Dick Barnes is a revered figure in the Commodore user community. Dick has written several articles for us, of course, but he is probably better known to Amigans as the editor of the late, lamented *The Amigan Apprentice and Journeyman*. This fine non-profit journal, published by Dick for three years, became renowned for tough, in-depth product reviews, fine tutorials and a wealth of useful, hard-to-get information. Dick presided over all with style, his pungent, pithy prose putting the lie to the old cliché that engineers can't write. Unfortunately for us all, Dick has suffered ill health recently, and was forced to slow down. Not surprisingly, no-one could fill his shoes on the masthead and *The Amigan* was retired.

So it's a special pleasure to see that, like an old friend come for a visit, Dick has published the *Final Supplement* to the *A & J*. In a recent letter, he wrote to say that it's a full 60-side issue that has taken him almost three months to complete.

Dick goes on to expound upon one of his favourite subjects:

> "Having just finished reading the latest copies of *TransAmi, Amiga Sentry, AmigaWorld, Personal Publishing*, and *PUBLISH!*, I'm in a vile mood about the continuing massacre of English in all of them – especially after *Personal Publishing* said, 'Goudy is one of the more classier text faces.' I think perhaps that the magazine needs a more smarter proofreader, right?
>
> "Also abundantly evident is this fact: No writers (and no editors, either) seem to know how to deal with the feminist-impelled substitution of 'their' for 'his' in sentences such as: 'Everyone has *their* own way of describing...', which is illiterate cow puckey and easy to evade. When I saw this blooper on page 40 of the July *TransAmi* (see Section 8), I damned near dropped my new uppers into the porcelainware disposal sub-system of my sanitary environment...
>
> "If I ever again see in *TransAmi* a sentence beginning 'Everyone has *their*...', I'm going to send you a copy of *Strunk on Style*, soaked for 40 days and night in the essence of skunk, and titled in dribbling red ink 'Another Massacre!' "

We are properly chastised. I am indeed one of those dread feminists who believe that sexism in language should be expunged, but, as Dick points out later in his letter, the correct way to have done so, in this case, would have been for me to change the offending sentence to begin "*All writers* have their own way of describing..."

Dick is the first to admit that he is not a perfect writer either. He concludes his missive with the following: "Whatever thy recent sins, I don't recant my opinion that Tim Grantham did a first-rate editing job on my last article. No one can edit his own stuff well. Another, good editor is required to make an article glisten. Tim did so, and I thank both you and Allah for his skill."

Coming from Dick, that's high praise. I've blushed every one of the several dozen times that I've read it. It would be most ungracious of me then to point out that "No writers (and no editors, either) seem to know..." should more properly be written "Neither writers nor editors seem to know..." – so I won't.

Dick likes to refer to himself as a curmudgeon. We know better. We know he's one of a rare and disappearing breed: a gentleman and a scholar.

We hope that Dick recovers his health soon. We can't wait to print another of his erudite articles.

**Tim Grantham,**
**Editor.**

# InterComputing Inc.

2100 N. Hwy 360, Suite 2101
Grand Prairie, TX 75050-1015

*In business since 1984; Amiga support since 1985*
*Member of Dallas Better Business Bureau*

We are a service oriented business offering discount prices

FAX: 1-214-660-3695

Customer Service and Order Status call: 1-214-988-3500

COMPLETE PRODUCT LISTING AVAILABLE

Call toll free: **1-800-622-9177**

# MEMORY EXPANSION

## Amiga 500 & 1000

| | | |
|---|---|---|
| Starboard 0 K | A1000 | 239.95 |
| Starboard 1 MB | A1000 | 499.95 |
| Starboard 2 MB | A1000 | 795.00 |
| Starboard 512 K | A1000 | 399.95 |
| Starboard Multifunction Card | | 74.95 |
| Starboard SCSI Module | | 99.95 |
| Starboard Upper Deck | | 44.95 |
| M501 0 K | | 59.95 |
| M501 512 K with Clock | | 159.95 |
| Mini Meg A1000 with 0.5 MB | | 359.95 |
| Mini Meg A1000 with 1.0 MB | | 499.95 |
| Mini Meg A1000 with 2.0 MB | | 789.95 |

## Amiga 2000 & 2500

| | | |
|---|---|---|
| 8-UP Memorycard DIP | 0K | 159.95 |
| 8-UP Memorycard DIP | 2MB | 595.00 |
| 8-UP Memorycard DIP | 4MB | 995.00 |
| 8-UP Memorycard DIP | 6MB | 1335.00 |
| 8-UP Memorycard DIP | 8MB | 1795.00 |

### VIDEO SOFTWARE

| | |
|---|---|
| Artistic Expressions Vol. I & II each | 39.95 |
| Broadcast Titler | 199.95 |
| Comic Setter | 66.95 |
| Deluxe Paint 3.0 | 109.95 |
| Deluxe Productions | 144.95 |
| Director | 44.95 |
| Director Toolkit | 27.95 |
| Fantavision | 39.95 |
| Innvision | 169.95 |
| Intro CAD 2.0 | 49.95 |
| Movie Setter | 69.95 |
| Pageflipper Plus FX | 99.95 |
| Pagerender 3-D | 109.95 |
| Pagestream | 139.95 |
| Pagestream Fonts #1-5 each | 29.95 |
| Photon Paint 2.0 | 99.95 |
| Professional Page | 239.95 |
| Sculpt 3D | 69.95 |
| Sculpt 4D | 399.95 |
| TV Show NTSC/PAL | 62.95 |
| TV-Text | 62.95 |
| Turbo Print | 32.95 |
| Turbo Silver | 129.95 |
| Zoetrope | 99.95 |
| Inovatools I | |

### MUSIC & AUDIO

| | |
|---|---|
| Audiomaster II | 69.95 |
| CMI Midi Interface | 69.95 |
| Dr. T's 4-OP Deluxe | 99.95 |
| Dr. T's Copyist | 199.95 |
| Dr, T's DX Heaven | 99.95 |
| Dr. T's KCS 1.6 | 166.95 |
| Dr. T's Kawai K1 Lib. | 99.95 |
| Dr. T's Midi Recording Studio | 44.95 |
| ECE Midi-A500/2000 | 44.95 |
| ECE Midi-A1000 | 44.95 |
| "M" | 154.95 |
| Midi Cable 5' black | 8.95 |
| Midi Cable 8' color | 11.95 |
| Midi Gold external | 69.95 |
| Midi Gold internal | 69.95 |
| Music Mouse | 49.95 |
| Perfect Sound | 64.95 |
| Soundscape | |
| Audiodigitizer | 79.95 |
| Pro Midi Studio | 124.95 |
| Utilities I | 39.95 |
| Studio Magic | 69.95 |
| Synthia | 66.95 |
| Texture | |

### BUSINESS

| | |
|---|---|
| Analyze 2.0 | 62.95 |
| BEST Management System 2.0 | 299.95 |
| Critic's Choice | 159.95 |
| Cygnus Ed Prof. | 69.95 |
| Excellence | 179.95 |
| Financial Plus | 199.95 |
| Kindwords | 66.95 |
| Maxi Plan 500 | 99.95 |
| Maxi Plan Plus | 129.95 |
| Microfiche Filer | 69.95 |
| Microfiche Filer Plus | 112.95 |
| Money Mentor 2.0 | 59.95 |
| Pen Pal | 109.95 |
| Phasar 3.0 | 66.95 |
| Prowrite 2.0 | 72.95 |
| Super Base Personal II | 99.95 |
| Super Base Prof. 3.0 | 199.95 |
| Superplan | 109.95 |
| TexEd Plus | 54.95 |
| The Works-Platinum | 169.95 |
| Wordperfect | |
| Shakespeare | |

### HARD DRIVES

| | |
|---|---|
| GVP Impact Drives: | |
| 0MB-80meg Quantum | 1199.95 |
| 0MB-No Drive | 299.95 |
| 2MB-80meg Quantum | 1699.95 |
| 2MB-No Drive | 699.95 |
| Hardframe DMA SCSI Controller | 239.95 |
| Phoenix Drives: | |
| 22Meg A-1000 | 599.95 |
| 22Meg A-2000 | 549.95 |
| 32Meg A-1000 | 699.95 |
| 32Meg A-2000 | 749.95 |
| 48Meg A-1000 | 799.95 |
| 48Meg A-2000 | 749.95 |
| 80Meg A-1000 | 1299.95 |
| 80Meg A-2000 | 1249.95 |
| Quantum Pro-80Meg | 875.95 |
| Supra 30MB Drive | 699.95 |
| Supra 30MB HDisk | 699.95 |
| Miniscribe | |

**FDATA-10 DRIVE**

Quiet, reliable drive for Amiga 500

$ 139.95

**FDATA-20 DRIVE**

Two 3.5" drives in one small case. Clean design, quiet, and at a great price.

$ 249.95

**HEWLETT PACKARD INK-JET PRINTERS**

HP-Deskjet 799.00
HP-Paintjet 999.00

### PROGRAMMING

| | |
|---|---|
| A/C Basic Compiler | 129.95 |
| A/C Fortran-AB Soft | 179.95 |
| Aztec C Developer | 219.95 |
| Aztec C Professional | 159.95 |
| Aztec Source Level Debugger | 59.95 |
| GFA Basic 3.0 | 99.95 |
| Lattice C 5.0 | 219.95 |
| Lattice C ++ | 399.95 |
| Lint | 69.95 |
| Power Windows 2.5 | 56.95 |

### MISCELLANEOUS

| | |
|---|---|
| Arexx | 32.95 |
| BAD | 32.95 |
| CLI Mate | 27.95 |
| DOS-2-DOS | 42.95 |
| Disk Mechanic | 56.95 |
| EZ-Backup | 32.95 |
| Facc II | 22.95 |
| Project D | 32.95 |
| Quarterback | 44.95 |

### VIDEO HARDWARE

| | |
|---|---|
| A-Pro Draw 9x6 | 399.95 |
| A-Pro Draw 12x12 | 499.95 |
| Amiga Live A 1000 | 219.95 |
| Amiga Live A 2000 | 329.95 |
| Amiga Live A 500 | 299.95 |
| Color Splitter | 89.95 |
| DigiDroid | 64.95 |
| Digi View 3.0 | 139.95 |
| Digi View Gold | 139.95 |
| Easyl 500 | 319.95 |
| Easyl 1000 | 339.95 |
| Easyl 2000 | 349.95 |
| Flicker Fixer | 499.95 |
| Panasonic 1410 Camera Kit | 239.95 |
| Perfect Vision | 175.95 |
| Polaroid Palette | 2199.95 |
| Super Gen | 699.95 |
| X-Specs 3D | 99.95 |

### DISK DRIVES

| | |
|---|---|
| A2010 Internal 3.5" | 149.95 |
| AIR Drive External | 169.95 |
| AIR Drive Internal | 134.95 |
| California Access Drive CA-880 | 169.95 |
| F-DATA 10 | 139.95 |
| F-DATA 20 | 249.95 |
| Drive Extension Cable | 19.95 |

### MODEMS

| | |
|---|---|
| Supra 2400 baud ext. | 149.95 |
| Supra 2400 baud int. | 149.95 |

## AMIGA 500 Starter Pack

A500 with 1Meg
2 Drives
Monochrome Monitor

949.95

Prepay with check or money order and get free UPS-ground shipping and a FREE GAME!

## A-MAX 134.95

The 'MAC' Emulator for the Amiga works with all Amiga models

## ROM-Set 139.95

For use with A-MAX

## 3.5" Drive 199.95

MAC-compatible; for use with A-MAX

## TRADE UP!

We will trade in your Amiga 500 or 1000 towards a new Amiga 2000 or 2500. Call for details!

Government, corporate & institutional

**Purchase Orders are welcome!**

*As always we have the most 'customer friendly' terms: S/H $3.95 in cont. USA; Minimum order is $30.00; MASTERCARD & VISA with NO credit card fee; in Texas add 7% Sales Tax. We ship to APO addresses, RMA# required on all returns. 18% restocking charge. All prices subject to change without notice.*

# Make Amigos With Other Amigas.

The largest group of Amiga® users in the world shares its problems and solutions online every day in CompuServe's Amiga Forums. And you can join them.

Whether you're an Amiga novice or a professional user in broadcasting, film special effects, animation, or music production, you'll find support from thousands of Amiga users and nearly every third-party Amiga software and hardware vendor.

Looking for a solid CAD program? Want to make the most of your Amiga's multitasking capabilities? Ask somebody who's been through it all. There's no better way to get more out of your Amiga.

To join CompuServe, see your computer dealer. To order direct or for more information, call 800 848-8199. If you're already a member, type GO AMIGA at any ! prompt.

**CompuServe®**

# LeTteRs

*Mr. Bob Parker wrote in to inform us:*

A Letters Column is sorely missed, since it gives a personal forum to the readers to interact with the magazine, as well as a less intimidating place for the readers to submit that little bit of code they're so proud of.

*Your point is well-taken, Mr. Parker. From this moment on, in order to accomodate the inclusion of letters, our regular feature articles will be postponed, shortened or just plain thrown out!*

*Seriously though, we agree with you concerning the importance of the Letters Column. You're quite right to contend that Letters is indispensible. You can look for it as a regular, nary to be missed feature from now on.*

**I wish to address this letter** to Ben Blish: *A Matter of Style*, Volume 2, Issue 2.

I must take exception to the declaration of GetMsg() as returning a pointer to an IntuiMessage structure, as it raises a maintenance issue. GetMsg() is an EXEC call that returns a pointer to a Message structure, and it should remain so.

Here's an example of how mis-declaring Amiga functions could lead to problems. Let's say we need to add a function to the same file in which Menu-Handler resides. In this function, we wish to assign the return from GetMsg() to a pointer to a plain old EXEC Message. Because of the redeclaration of GetMsg(), we would get a compiler error unless we cast GetMsg() as a pointer to an IntuiMessage! This state of affairs is clearly unacceptable. The correct method is to **#include** prototype files for all Amiga functions used and to cast as necessay. It doesn't hurt to perform a necessary cast or two, and it might avoid problems.

While I'm up here on this soap box, I would like to address your rule on assembly code in C programs. It's fine as far as it goes, but it lacks a certain emphasis. I would rephrase it as follows:

**Law**: Anyone who uses in-line assembly code in a serious C program will be drawn and quartered.

I feel the punishment fits the crime most appropriately.

Walter Strickler, Boulder CO.

**I have just received** Volume 2, Issue 2, and I find myself snickering a little over the letters. In this issue, I think Larry Phillips has hit the nail on the head about AmigaBASIC – and I also expect Commodore to ignore the whole thing.

So far (about six years), I have seen no evidence that Commodore has any competence at all outside of their Engineering Department(s). Why would anyone expect them to pay attention to anything that makes so much sense?

The problem with most MS-DOS ports is not their origin, but that those who do the porting don't think much beyond MS-DOS. As evidence that's not always true, consider WordPerfect.

I had the opportunity recently to compare WP on the PC to WP on the Amiga. And, it's much nicer to use on the Amiga! Of course, that's because those who handled the port didn't restrict themselves to the PC interface, but added all the Amiga bells and whistles. It is interesting to examine what has happened.

I have been following the dialogue concerning the product. First off, it appears that Amiga WordPerfect sales have not been strong enough to justify continued Corporation support. Secondly, the forums have been filled with anguished howls over "bugs" which I didn't even notice.

One of them required a message from the developers of WP to explain it to me, and I'm still not sure that I see it. From my standpoint, WP does what I want, and without fuss. I can't help but contrast WP's behavior with that of Shakespeare. It crashes without warning when memory usage over-runs resources.

A CSA TURBO
68020
FOR ONLY
$399?

Hard to believe isn't it? CSA broke the price barrier in 32 bit technology. Now you don't have to settle for a far less capable 68000 accelerator, you *can* have affordability, capability, and speed in one easily installed package.

**"1000% faster than my stock Amiga™?"**

CSA's new 68020 *Midger Racer*™ flies through MaxiPlan™ spreadsheets over 60% faster than the standard Amiga™, and over 50% faster than 68000 based accelerators. With the addition of a 12 MHz 68881 co-processor, the *Midget Racer*™ can crunch 2 Million FLOPS over 1000% faster than a stock Amiga™.

**"600% faster than a 68000 accelerator?"**

In fact, programs like Sculpt & Animate 3D or 4D™, Draw 2000™, and X-CAD™ have been written to directly access the 68020 & 68881, and may not even *run* with a 68000 based accelerator. And with a 12 MHz 68881, CSA's *Midget Racer*™ is up to 600% faster than a 68000/68881!

**"All that for only $399?"**

No, there's more! It's also easily installed into ***any*** Amiga, supports an '881/2 up to 33 MHz, and it's available now. Don't settle for second best, call your Amiga dealer or CSA today!

CSA. We set the standards in performance.

CSA

Computer System Associates, Inc.
7564 Trade Street
San Diego, CA 92121
619-566-3911 FAX: 619-566-0581

Trademarks: Amiga: Commodore-Amiga, Inc.; MaxiPlan: Oxxi, Inc.; 68000/020/881/882: Motorola, Inc.; Sculpt & Animate 3D/4D: Byte by Byte, Inc.; Draw 2000: Aegis, Inc.; X-CAD: Taurus, Inc.; Midget Racer: Computer System Associates, Inc.

For Jim Butterfield: **bss** stood for "block starting with symbol" in the old 709 assembler (I think it was called SAP, for Symbolic Assembly Program). As far as I know, that's where it got started. There was also a **bes**: block ending with symbol. The 709 index registers only decremented, and working backward through arrays was sometimes more convenient.

J. L. Schilling, San Diego, CA

**The exchange of letters** in the May column of *Transactor for the Amiga* was interesting. But with all that was written, only the tip of the iceberg (remember the Titanic?) was explored.

I'm a technical engineer using a dual-floppy Amiga 2000 as my home machine. I've had it for about six months. I work with PCs, a VAX, and IBM mainframes at work. It was my experience with advanced (real) operating systems that led me to purchase an Amiga instead of an MS-DOS compatible system.

Calling all user groups!

**Did you know that you can get volume discounts on subscriptions to *Transactor for the Amiga*?**

**For more information, call:**

**(416) 764-5273**

**or write:**

**Subscriptions Dept.**
***Transactor for the Amiga***
**85 West Wilmot St.**
**Richmond Hill, ON**
**Canada. L4B 1K7**

**Act *now*!**

Mr. Wright's complaint about "half-baked productivity software" is valid. IBM-PC business software has a certain 'industrial strength' feel to it lacking from most Amiga software. The shame of this is that the Amiga can keep up with any PC in existence, including the highly-touted '386s.

But it seems that the operating system, much more powerful than anything ever dreamed of by MicroSoft, is so complicated that developers are still climbing a steep learning curve. Four years after the introduction of the system, we still have commercial software that crashes on insufficient memory.

With regard to professional products, as I browse the latest issue of *IEEE Spectrum*, I see adds for the following software products:

- *Simscript*: a high-powered simulation program with animation graphics to make your simulations "come alive".
- *Signal Processing Workstation*: a graphic interface program supporting the design, analysis, and simulation of digital signal processing systems.
- *MathLAB*: an interactive math computation system with 2D and 3D graphics claiming to calculate a 1024-point FFT in 0.16 seconds on a '386 PC.
- *Maxell*: a specialized CAD package for the design, analysis, and simulation of electromagnetic systems.

I think you get the point. This is true professional software, and it is not available for the Amiga. Could the Amiga support these kinds of applications? Hell, yes! Especially with an Amiga 2500. The Amiga's high-speed windowing and multitasking operating system makes it a natural for these kinds of products.

But the research for this type of software may occupy a team of developers with Ph.Ds for a year. From whom in the Amiga developer community can we expect this effort and these products? Electronic Arts (and I'm not trying to pick on them unfairly)?

The Amiga will be perceived as a professional machine when most

professions have access to software of the above calibre. The Amiga magazines could help by seeking out small companies who are trying to write these products and give them some exposure.

Indeed, most Amiga magazines devote three-quaters of their review space to games. Browse *PC World* magazine and count the number of game reviews in it. This is the type of publication needed for professional acceptance of the Amiga.

As I said above, the shame is that the Amiga is capable of performing all these tasks. Remember that IBM entrenched themselves in the corporate computing world with a 5 MHz 8088 and a "souped-up CP/M". Since then most of the improvements have been to the hardware. MS-DOS is just as underpowered as it ever was.

And there isn't an operating system available for the PC at any price that can multitask better than AmigaDOS and Intuition. We have the hardware now. Let's get some real productivity software and tell the world about it. It can be done.

Tab Trepagnier, Kenner, LA.

**Bravo to Ben Blish** and Timothy Keating for persisting against short-sighted results. I admit I'm opinionated, but I have spent too much of my life deciphering poorly written code (including my own) not to applaud the spirit and substance of their article, *A Matter of Style*.

I started coding when math teachers were given a FORTRAN manual and told to train us. But even then, I sensed there was somethng wrong with calling an exponential routine to alternate the sign of terms of an infinite series. It wasn't until then that I learned structured programming techniques. Now, I proclaim them religiously.

We always say that every technique learned lengthens the rope with which we hang ourselves. In C, I see a fabulous power, but also, a hangman's dream come true. The short-cuts available are so appealing and so dangerous.

So, I have a couple of suggestions. These are some of my rules for structured languages.

- Include a closing statement for the **if**, **do**, **while** and **switch** constructs. For example, I include an **endif** after the closing curly-brace, but at the same indentation level as the if. Before reaching for K&R, please allow that I have placed the following statement at the top of my code:

```
#define endif /**/
```

  Consequently, the compiler ignores the **endif**, but the human reader is given a strong visual signal where the **if** clause ends. The other closing statements are **enddo**, **endwhile** and **endswitch**.
- When functions are closed with their name included as a comment, it is easier to determine where to add additional functions.
- I avoid the compound assignment statement (the **?** and **:** construct), the for loop, and combination operators (=+) because they hide the visual representation of the program's intent.

Representing the computer's task is programming. We can do better than the law profession. There, a whole industry exists only to interpret what the rest of the industry has written. We, however, can write in a manner that is comprehensible to some who need only understand the computer's task.

Gene Holmerud-Nye, Phoenix, AZ.

**A potential hardware bug** that may result in a high voltage discharge (snap) is prevalent in several early models of RGB monitors supplied for C128 and Amiga computers.

Since the advent of the Amiga and C128 computers, a small but significant number of us have experienced a problem with a sudden snap, crackle, or popping sound accompanied by a momentary loss of picture on the monitor.

This event would usually be infrequent to start with, but the snapping would become more frequent with time.

On one Amiga 2000 system, it developed immediately, and was so bad it would crash the system. I suggested that the owner return the system and have it replaced. I haven't seen the owner since, and I suspect he returned the whole Amiga system.

In the first year that the Amiga was on the market, I purchased my Amiga 1000 and the associated monitor at a Chapter 11 sale. The monitor occasionally cracked an electrical snap. About a year ago, it gave a snap that evoked a guru (system crash).

The snap seemed to go away for a few months. Later, when my computer system was used for a demonstration at a club meeting the electrical snap returned. Inquiries did not provide any answers.

I had the monitor checked over. We found taking the cover off caused the snapping to stop. A muffin fan was attached, but soon after, the computer developed an occasional lockup when I attempted to save or print a file.

The frequency of the lockup increased, and in short order, the computer could be used for less than an hour before it locked up. I observed the lockup occurred after there was a snap in the monitor.

Intermittent problems are the hardest to correct, but the problem had to be isolated and solved. Frank Gerard of ECS, an authorized Commodore service shop, worked on the problem with several different monitors and concluded that it was related to the red High Voltage wire going from the high-voltage transformer to the anode cap on the top of the picture tube.

He sprayed a plastic insulating coating around the high voltage transformer where the red wire emanated. This seemed to correct the problem in one or two monitors, but not in mine nor several others.

Bill Hanley, an engineer for the local public TV station, also brought in monitors with snapping problems. After a dis-

cussion with Frank, Bill checked several bulletin boards, and found information about a trace being established on the monitor mother board by arcing.

Frank studied the bottom side of the mother board and found evidence of carbon traces created by shorts from a heat shield tab to ground on several snapping monitors.

There is a heat sink for a voltage regulator on the side of the monitor near the high voltage transformer. This heat sink is a L-shaped metal plate about four inches high and five inches long. It angles at the rear corner for about an inch along the back of the monitor.

The heat sink is attached to the mother board of the monitor by two screws and a twist metal tab that extends through the mother board.

The metal twist tab is very close to the ground foil on the bottom side of the mother board. When twisted, it comes very close to the ground foil. And it doesn't take much voltage to bridge such a narrow gap.

This pathway is on the bottom side of the mother board and out of sight, so it seemed an improbable source for the problem.

We concluded that the large heat sink seems to act as a large capacitor near that high voltage wire. When voltage builds up high enough in the heat sink, it discharges with a small spark to the ground foil.

Consistent discharges inside a dirty monitor will help build a trace path to ground increasing the frequency and size of the voltage discharge. If the discharge is large enough, the high voltage on the ground trace will be reflected back to and through other monitor components and back to the computer. This problem seems to be prevalent on Commodore/Amiga models 2002, 1902 and 1080.

The Amiga 1084 monitor and other RGB Commodore monitors have a smaller heat shield and seem to have a better high-voltage cable path that is farther from possible conducting paths and seems to prevent the problem from occurring in those monitor models.

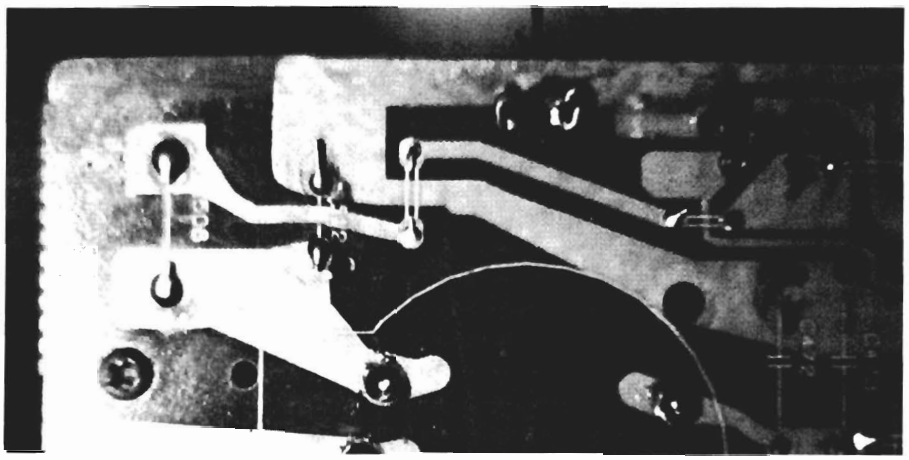
Four steps were taken to solve the problem *(please remember, even powered down, monitors can retain potenials of over 20,000 volts. Technical experience is a must. Don't say we didn't warn you! -Ed.)*:

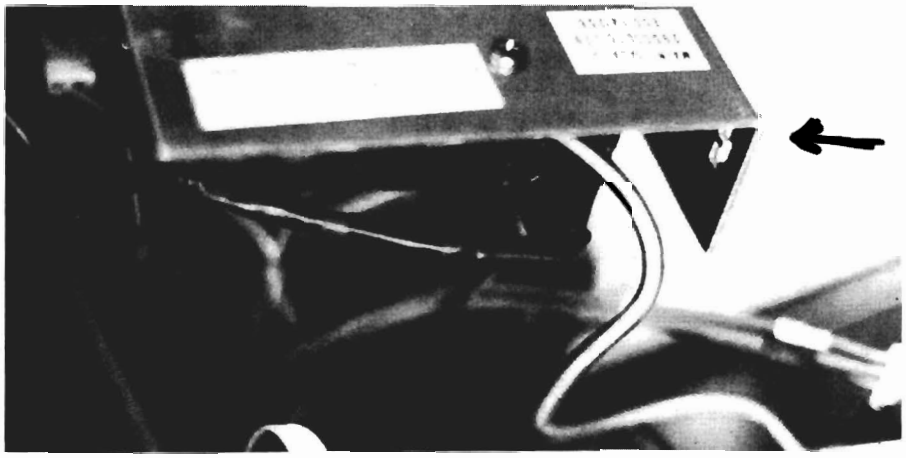
1. Clip the shorting metal twist tab from the heat sink. The two screws can hold the heat sink adequately. Carefully scrape any carbon traces on the mother board created by arcing from the tab slot to the ground trace.
2. Provide extra electrical insulation around the red HV wire. Split heat shrink tubing to wrap around the HV wire and use electrical tape to completely encase the wire with extra layers of insulation.
3. Spray Koloid Clear Acrylic plastic around the wire and the high voltage regulator to reduce leaking from high voltage sources.
4. Use TV Corona Dope to plastic coat area of twist tab slot and the ground trace.

Frank completes the repair at a cost of $30.00 US, but other service centers say they can't do the job for that price. A reasonable charge might fall into the $50.00 US range.

In two months of heavy use, no noticeable snapping has occurred.

Dick Landry, New Brighton, MN





**Mother Board View** (top): Left rear corner on the bottom side of the motherboard of a repaired monitor. The arrow points to ground foil and the heat sink tab slot.

**Heat Sink & HV Wire** (bottom): A view of the top of the L-shaped heat sink (arrow) and the red high-voltage wire. Note the extra insulation on the HV wire.

# Amiga Dispatches

## *Home computers and changes at Commodore*

**by Don Curtis**
*Copyright © 1989, Donald A. Curtis*

*Don Curtis is a police officer in Denver Colorado, who has for the last three years been assigned to program design and development along with system design and maintenance of ten AT&T Unix 3B2 computers. In his spare time, Don is assistant sysop on CompuServe's Amiga forum.*

### A perfect home computer?

My thoughts on the following aren't fully formed yet, so read this section as a rambling discussion that may end up being thought-provoking. Perhaps you have some ideas along the same line. Perhaps while reading this, you will come up with a few new ones. If so, great! That's the whole idea behind this discussion. If not, well, you may just find it interesting anyway, so read on.

I've been reading computer magazines for a long time, and for a long time, things seemed pretty stable. The only big change was that prices kept going down. Oh sure, every now and then this manufacturer or another would come out with a new machine. But generally, those machines were so similar to others already on the market that it really wasn't a big deal.

But now, I read about new computers coming out that offer significant speed and power improvements. These computers don't come out once a year. No, there seem to be new offerings three or four times a year. And, along with the new power and speed, comes a significantly higher price tag. It seems that a doubling in power brings a near doubling in price.

I like the power idea (don't we all want a more powerful computer sitting on our desk?), but I'm not so sure about the new prices. I have no problem with manufacturers being paid for their efforts. They have every right to a fair profit; that's what economics is all about.

There are two things that worry me, though. The first: the escalating cost of the home computer. And the second: will rising prices turn away the traditional home computer buyer? As prices go up, folk who could afford a computer in the past are priced out of the market. As the number of features increase, folk who don't upgrade their systems tend to feel they're left with an obsolete computer – especially after two or three new model releases/upgrades in a short period of time. There are a lot of reasons why they don't upgrade, but the obvious one is that they can't afford the new machine. Of course, there's another good reason: they're perfectly happy with their current equipment!

But what happens when they do decide they want to move up? Their machine is so 'old' (technology wise) that it's not worth very much in trade. The new machine is so 'new' (also technology wise) that it carries a premium price tag. The premium price tag keeps them from buying, and the fact that their computer is now an older model causes some to lose interest. In other words, we've lost a user.

And that brings us to the questions of how powerful should a personal computer be, and how does a 'personal computer' relate to a 'home computer'? Once, a personal computer was pretty easy to define. It used to be that a personal computer was one that you could sit on your desk, and you were the only one using it. You ran word processors, small databases and spreadsheets. It increased your productivity.

But now that so-called personal computers are hooked into networks, you may not be the only user. You're hooked into large file servers. Your personal computer acts as a front end to large banks of data. You can analyze more data and do it faster now than you could in the past. For the work environment that's great, productivity has increased again.

In the home environment though, we now have machines that are overpowered for typical home uses. You really don't need a machine that can do multiple millions of instructions per second (MIPS) or floating point operations per second (MFLOPS) for traditional 'home' uses (word processing, home accounting and games).

There are, however, specialized home uses where such power may be needed. If you are into doing complex ray-traced pictures, large complex calculations or any other heavy computational activity, then you would need all the power you can get.

You see where I'm going? There's now a blending where there used to be a distinction in the "home" vs "personal" area. The power of today's micro is reaching into the area that used to be reserved for the mini or in some cases, small mainframe! The distinctions are blurring, and the prices are rising. Again, I'm not sure what all this means, but I'm not so sure it bodes well for John (Jane) Average User.

Take it down to specifics and the Amiga is no exception. The A500 is not enough machine for many (myself included), but the A2000 in a reasonable configuration (A2000HD), the "perfect" personal computer for many, may be a bit too pricey for those same folk. It is relatively easy to justify a near $1,000 computer for use in the home. Moving into the near $2,000 area, well....you could stretch your budget that far. But when the cost starts moving into the near $2,500 and up range, you've got to have an awfully good reason (or a lot of money to play with) to justify such an expenditure.

In my case, that was exactly the problem. I didn't get everything I would have liked with my A2000. I would have liked, and could have used an extra 2 Meg of memory. And even though my dealer gave me a healthy discount on my computer, I had a limit on what I could spend. Adding that 2 Meg of RAM put me over the limit.

*Commodore has announced new products that should be available in the Fall of 1989...*

Lately, a mini A2000, perhaps called a A1500, has been the topic of much discussion on some of the networks. This computer would be a separated system unit/keyboard like the A2000. It would have a few expansion slots (perhaps 3 horizontal mount slots rather than the current 5 vertical mount), no CPU slot, no MS-DOS slots, a single internal 3.5" drive and a spot to mount a 3.5" HD but no 5.25" drive bay. It would still have the video slots, but no composite output. A machine like this could probably be sold in the $1,400 price range. It would have the expandability the A500 doesn't have, but not all the expandability the A2000 has, nor would it have all the bells and whistles.

With no side expansion slot, you'd be pretty much limited to the 3 internal slots. No BridgeBoard slots (that would be a plus for many), but it'd still be more useful for video applications than the A500 due to the video slot. No CPU slot would mean no A2620-type cards, but someone would probably come up with a CPU replacement board if you really had to have a more powerful processor.

All in all, it might just be the perfect computer for in home use. It wouldn't have helped me, I had to have the BridgeBoard, but for many folk, it would have made the choice between the Amiga and some other brand of computer much easier.

And being so similar to the A2000, it may have helped convince many current A2000 owners that their machine is still "up to date". Even if they haven't moved up to an A2500 or beyond, they'd still feel they weren't left out. Best of all, this A1500 would cost less than the current A2000! In fact, if such a machine existed, I'd bet it'd soon start to outsell the A500.

### What's happening at Commodore?

As you should all know by now, Max Toy is out as the head of Commodore US (CBM) and Henry Copperman is now in charge. As you should also know, marketing efforts in the United States have been poor to say the least. Well, Mr. Copperman quickly took charge and has big plans to change all of that.

Lloyd Mahaffey, formerly of Apple (as was Mr. Copperman) has been named Vice President of Marketing. Robert Larsen, formerly of Casio, has been named Vice President of Consumer Product sales. Commodore Amiga Technical Support (CATS), headed by Gail Wellington has been upgraded from part of the Research and Development organization to division status reporting directly to Mr. Copperman.

That's not all. Worldwide, Commodore International (CBU) is making large changes in their management. Willy Scheiwiller, formerly with Industrade, the exclusive distributor of Apple Computer products in Switzerland, has been named general manager of Commodore International's Swiss operation. Peter Kaiser, formerly with Compaq Computer has been named sales and marketing manager for Commodore Amiga in Germany.

Klaus Dalheimer, who had been with Hewlett Packard, has been named general manager in Austria. Erik Schale, who came from Apple, takes the top spot in Scandinavia. Ex-Olivetti employee Bernard Van Tienen takes charge of Benelux. Peter Baley, from Compaq, now is the Vice President of International Marketing.

Pat Byrne, who had been with NEC, now will be general manager of the Australia/Far East sector. Jaime Figueroa, formerly with IBM, has become the Director of Sales for Latin America and finally, Brian Weyman, formerly of Xerox, is now Vice President in charge of WorldWide Manufacturing and Purchasing.

Back to Commodore Amiga here in the US: CATS is hiring more people. They've posted a notice on UseNet looking for Support Programmers (multiple categories), Assistant Technical Editors, Administrators, Applications Support people, Marketing Support people and others. Rumor has it they'll be doubling their size.

Gail Wellington intends to form a developer's advisory board, establish a world wide communications link so developers world wide can have direct contact with CATS. They're going to add marketing support, try to coordinate development of us-

er interfaces so all applications "feel" the same, establish documentation standards and try to take a more active role in key third-party products.

All this is intended to increase market penetration and diversity. The focus is going to be on the government, education and multi-media markets. Lloyd Mahaffey is making no bones about the Federal market. He has said that in six months you'll see Commodore entering the Federal market with some impressive products. Since Mr. Mahaffey had in the past been given credit for Apple's visibility in the Federal market, his statement should be taken with much more than a grain of salt.

He'll probably pull it off. Mr. Mahaffey has also been quoted as saying, "When I took a close look at the Amiga, I couldn't believe it. The machine has full multi-tasking, a Motorola Inc. 68000 processor and Icons. What it doesn't have is marketing and breadth of applications. The product has never been positioned right and the company never evangelized the software developers".

To further enhance that statement, in addition to the other changes, new products will now be market driven, and not product driven. That is, you now get what you want, not what someone else thinks you want. Product development is going to be accelerated (new employees noted above), and production yields are going to be increased. As an example of that, the market for networks is growing rapidly. Commodore needed a network solution so it bought out Ameristar, the long time sole source for Amiga network boards. They are apparently also working very closely with Novell on network applications.

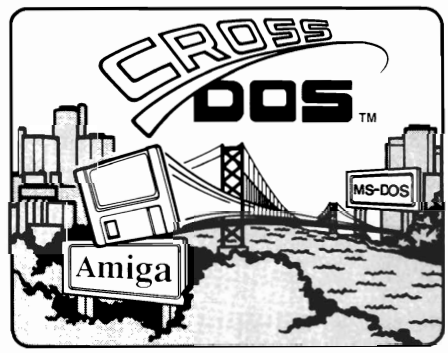
Another major change is that advertising is going to increase. From 1.1 million dollars in fiscal 1988 (July 87 to June 88) to 16 Million dollars in fiscal 1989. Much of that will be starting about the time you read this in anticipation of Christmas sales. The intent is for every person in the target market to see an Amiga commercial an average of 20 times during October, November and December!

In fact, presuming Commodore does spend this money on advertising, that amount will place Commodore in the top 80 advertisers in the US. IBM and Apple don't spend enough to be on that list, but how many times have you seen the Little Helo commercial from Apple? What about the old M*A*S*H crowd commercial for IBM? If all goes well, you'll see Amiga commercials more often.

The commercials, a 60 and a 30 second spot, will be produced by George Lucas and directed by Mathew Robbins. Rumor has it that George Lucas turned down several other commercial jobs, but has an Amiga and likes it so much, he decided to do the work for Commodore.

Good news for dealers also! They've gone after the Grey Market in a big way. Sources within Commodore report that Commodore has found the dealer supplying one of the major Grey Market mail order operations and shut them down. Not

. . . if you have at least 3 years of assembly language programming experience, preferably one year of 68000 or have an NTSC/PAL Video Hardware background. Join the team that makes no compromises and help us change the world. Key positions available at the fastest growing Amiga company. Hackers call Stephanie at 913-354-1146 or write to Stephanie at 115 West Crane Street, Topeka KS 66603.




**the MS-DOS File System for the COMMODORE AMIGA™**

AVAILABLE NOW FOR THE LOW PRICE OF

**$30.00 (US)**
**$36.00 (CDN)**

**CROSSDOS™. . . MS-DOS® DISK ACCESS DONE RIGHT!**

**The first MS-DOS File System for the Amiga®**

- Reads or writes any 360K or 720K MS-DOS or ATARI ST® disks (Version 2.0 or higher) with optional text file filters.
- Transparently accesses MS-DOS files from any utility or application (including file requesters).
- Fully integrates itself into the Amiga operating system.
- Automatically readjusts to different MS-DOS/ATARI ST formats.
- Can be removed after use to reclaim memory.
- Provides an easy installation program.
- Availiable in a READ-ONLY version from the Public Domain or directly from **CONSULTRON** for only **$5.00.**

For orders placed through **CONSULTRON** add $3.00 shipping and handling ($8.00 outside the U.S. and Canada). Michigan residents must add the correct sales tax. C.O.D. add $3.00

**IN ADDITION** to reading and writing any file on an MS-DOS disk, perform the following DOS functions on files and directories.

- Scan any directory
- Create directories
- Rename
- Delete
- Set dates
- Set protection bits
- Seek file positions
- Get disk information
- Add cache buffers

Send check or money order to:

Please allow up to 2 weeks to process your order.

**Dealer inquiries welcome.**

**CONSULTRON**
11280 Parkview
Plymouth, MI 48170
Technical Support
(313) 459 - 7271

**Amiga** is a registered trademark of Commodore-Amiga, Inc. **MS-DOS** is a registered trademark of Microsoft, Inc. **Atari ST** is a registered trademark of Atari, Corp.

only that, but Commodore has not only put in place more checks to prevent future re-occurrences, but has sent a letter to all their dealers warning them of the consequences of Grey Market dealing.

For those of you who don't know what a Grey Market operation is, let me explain. An authorized dealer will buy large overstocks of product at a discount from normal dealer prices (thru bonuses, special promotions, etc.) and then sell them to an un-authorized dealer.

Un-authorized dealers also buy product from authorized dealer's who need quick cash or are going out of business and have remaining stock at "fire sale" prices. The un-authorized dealer will then sell to the public (normally through mail order) at prices that authorized dealers can't meet.

Dealer support is being increased, and all dealers will be put through a re-evaluation of their qualifications and minimum purchase requirements. Supposedly all dealers will soon have minimum requirements of $200,000 worth of product per year. While that will weed out the smaller dealers (some of whom may be doing a good job anyway), it should insure that the remaining dealers are firmly committed to the Amiga.

The Amiga A500 will more than likely be moved into the non-discount mass market area. That could include stores like Sears, but would certainly exclude stores like Federated and Toys-R-Us.

And finally (as if the above isn't enough), Commodore has announced new products that should be available in the Fall of 1989. Those products are:

- A2630/68030 processor board (like the A2620) that runs at 25 MHz and includes a 25Mhz 68882 FPU and will hold 2-4 Meg of 32 bit RAM.
- A2091 Autoboot HD controller (the one used in the A590) that will support up to seven SCSI devices (but no ST-506 drives directly) and also has sockets for up to 2 Meg of 16-bit RAM. This controller even works under conditions of severe overscan or other heavy DMA activity. You will be able to mount a 3.5" drive on the board itself.
- A2094A kit to bring an A2000 up to an A2000HD configuration. It includes an A2091, 40 Meg HD and all necessary cables and the drive will be factory configured.
- The A2090B is the long promised half size card that will allow the A2090 to autoboot under OS v1.3 and above.
- A2232 Multi-serial port card. It will have seven serial ports on a single card and includes drivers for both UNIX and AmigaDOS.
- A206510 Megabit/Sec Ethernet board that supports both Thick and Thin Ethernet standards. It will come with AmigaDOS support for NFS, TCP/IP and Novell standards.
- A2024 Hi Resolution monochrome monitor that supports upto 1008x800 resolutions and has "AutoSwitch" to lower resolutions and de-interlacing. It will require 1 Meg of RAM and will work on any Amiga.

No prices have been given for the above, but by the time you read this pricing should have been set (and perhaps some of the above on your dealer's shelves).

As you can see, Commodore has some very big plans, and a very big job ahead of them. Let's hope that it's not all just talk or in vain.

This is the shot in the arm that the Amiga has needed all along. With the new management in place, we may very well see it all come true. If it does, the Amiga will take its rightful place among the top-selling lines of personal computers.

### Good news for A1000 owners!

I've even got some good news for owners of the A1000. Greg Tibbs, a private hardware developer, has developed the *Rejuvenator*, a board that will allow the A1000 to use the new Enhanced Chip Set (ECS) including the new 1 Meg Agnus. While others have been saying that it's not possible to upgrade the A1000 to the ECS, Gregg has been working on doing just that and apparently has done it!

The Rejuvenator will replace the present Kickstart WCS board in the A1000 and all the custom chips will be moved up onto the new board. Very little soldering will be required and all chips on the Rejuvenator will be socketed. Folk who have seen a prototype state that it will be "a snap" to install.

You pull a few chips from the motherboard, put them on the new board. Pull the Kickstart WCS board and push the new board on the old connectors. You'll perhaps need to make three additional connections to the motherboard. No other modification will be required, that is, you will not have to cut any traces or unsolder any chips.

Once you make the change, you'll have KickStart in ROM (or you can elect to keep KickStart on disk if you so choose..switch selectable), 1 Meg of CHIP RAM, memory expanded by up to 1 Meg (less if you choose to keep KickStart on disk). Also included is a battery-backed clock/calendar and a connector for a Flicker Fixer.

Expected price for the board and chip kit is $500. No ordering information is available at this time, but Greg can be contacted on CompuServe in the Amiga Tech Forum (GO AMIGATECH). Greg's PPN is 73767,456.

Greg also posed an interesting question to all about another project he is working on. He has an idea for a cheap accelerator that would increase the CPU speed to 10 MHz (it's now 7.16 MHz, so that'd be about a 30% speed up) and solve the flicker problem for about $35! He says it will require a multi-sync monitor. That's all I know about this project for now, but it sure sounds interesting!

That's it for this issue, things have been hectic lately, but they sure look good for now. Only time will tell, and we've got lots of that. Right?

# ACCESS

## *A scripting language for the Amiga*

**by Steven Ahlstrom**
*Copyright © 1989 Steven Ahlstrom*

*As the primary sysop on CompuServe's Amiga Forum, and a veteran Amiga programmer, Steve Ahlstrom sees just about every notable public domain and shareware program that gets written, and in this column he tells you about the ones that have recently caught this eye. If you have some information that you would like to share through this column, you can leave a message to Steve on CompuServe (his PPN is 76703,2006), or send the info to us, and we'll pass it along.*

ARexx is probably the most significant piece of Amiga software released to date. It allows programs with an ARexx port to communicate with and control other programs with ARexx ports. This ability allows for an integrated and cohesive system. With ARexx you have the ability to integrate programs from various software manufacturers into a personal environment that meets your needs.

The entire Amiga community has seen the potential and power of ARexx. Even Commodore themselves recognize a good thing (no, a *great* thing!) when they see it. ARexx will be a part of the next OS upgrade (1.4). (ARexx is currently a commercial product, $49.95, by William Hawes)

Now, from Khalid Aldoseri, is another product that goes hand in hand with ARexx – *Scripit*. Scripit has the same staggering potential as ARexx. It's not an ARexx clone, but... well, read on.

Scripit is, as the name suggests, a scripting language for the Amiga. It can do practically anything you would normally do with the mouse and keyboard, but via a script. It is a script processor that reads in a script file and executes all commands in that file sequentially. Scripit can control any program by activating its menus and gadgets, as well as being able to click anywhere within the program, just like ARexx does, *but*, without the need for a special port.

Scripit can control programs that ARexx cannot. Scripit is not a programming language like ARexx. However, Scripit does have an ARexx port. But Scriptit does have an ARexx port., so it can be fully driven from ARexx, giving ARexx control over programs that don't have ARexx ports.

Scripit is a set of programs – a *Recorder*, a *Script Player*, a *Lister*, a *Workbench Interfacer*, and some assorted programming aids. The Workbench interfacer hooks into the Workbench and allows Scripit to fully control Workbench. You can select icons, open and close drawers, move icons, run programs, everything you can do with the mouse, but from a script. You can reference icons by name. There is no need for trying to figure out screen coordinates.

Scripit has extensive window/screen manipulative features. You can move, resize, close, and depth arranges windows and screens. A full set of console commands are provided as a means to open and close consoles and send virtually any ANSI command to those consoles.

The command set is very rich. It has a set of graphics commands to allow drawing on any program's screen or type text into any window.

Scripit can be run from Workbench or the CLI. When double clicked from Workbench, a file requester appears. It will initially search for SCRIPIT:. If that is not found, it defaults to the current directory. From there, just click on the script you want to execute. You may also create an icon for scripts (any project icon will do) and execute the script by double clicking on it.

Scriptit has several command line options, they are:

```
Format: Scripit [flags] [scriptfile]

Flags:  -v Turns on Verbose mode.
        -d Turns on Debug mode.
        -x Loads Scripit Resident Arexx mode.
           Format: Scripit -x [portname]
              Where PortName is the portname Scripit
              will be referenced by from ARexx.
        -c Compile Script.
           Format: Scripit -c [input file]
              [output file]
        -w Disable Warnings.
```

Scripit has the ability to compile scripts. This will give a 30%-50% reduction of needed disk space for the script. The program XIT (included in the distribution Scripit release) is a special version of Scripit. XIT will only run compiled scripts. XIT is much smaller than Scripit and, as such, loads quickly.

Should you attempt to execute a non-compiled script with XIT, it will abort. Compiled script files can be identified by the first four bytes in the script (TYPE scriptname OPT H). They are X, I, T, 1. When using Scripit to execute a non-compiled script, it will automatically compile the script then execute it.

Should Scripit not be able to read the entire script at once, it will compile and execute each line of the script a line at a time. There is also a 'live' mode. By typing **SCRIPIT ★** from the CLI you may enter commands directly. Each command will be executed by typing a return at the end of each line.

The Recorder is an Intuition event recorder. It records exactly what the user does and saves it to a file. The file it saves is itself a script. If executed from Scripit, it will recreate the recorded events. The Recorder is *not* just a keyboard and mouse move recorder. It records all messages going from Intuition to the currently active window. Should you change the active window while recording, the Recorder will add the necessary commands to change the active window as well.

While the Recorder is 'turned' on, the power LED will be flashing to let you know you are in record mode. After turning the Recorder off, you may edit the resultant file just like any other text file: add, delete commands, whatever you wish to do.

The Recorder allows for several options also. You may automatically add comments to its generated script file. You may have it record menu and gadget numbers rather than names if you like, or not have the power LED flash while the Recorder is active. You can disable time frame recording, MouseMoves, and IntuiTicks. The command line options are:

```
Format: Recorder [-flags] [scriptfile]

Flags:  -c Adds comments to script.
        -w Do not open display window.
        -l Disable power LED activity meter.
        -n Use menu & gadget numbers not text.
        -t Disable time recording.
        -m Disable MouseMoves recording.
        -i Disable Intuiticks recording.
```

The Lister is a general listing facility for screens, windows, menus and gadgets. It will help in identifying the names or numbers of the items to be controlled. Just run Lister. You will get a list of screens in the system.

You can see the windows in a specific screen by typing the first few letters of the screen name as Lister's first argument. You can go on to specify window, menu, menuitem, subitem, or gadget. The format is:

```
Format:  Lister  {screen] [window] [menu] [item]
                 [subitem]
or:      Lister  [screen] [window] -g [gadget]
```

XitLoadWB is the Workbench Interfacer. This program is what gives you full control over the Workbench. To use this feature you must change the "LoadWB" line in your startup-sequence to "XitLoadWB".

On the next reboot XitLoadWB will load the Workbench for you and inserts its hooks (XitLoadWB calls LoadWB). There are a variety of Workbench commands, Select, Open, Close, Deselect, ShiftSelect, ShiftOpen, SnapShot, CleanUp, Re-Draw, Info, Menu, Drag, DragTo, and DragOver.

Scripit has an "AutoScript" feature. You can construct a script to automatically run. Instead of typing (at the CLI) "Scripit scriptname", you only have to type "scriptname". This is similar to ARexx when used in conjunction with WShel:, the RX command isn't needed.

I realize I've thrown a lot of stuff at you, and by now you may be totally confused! So, let's look at some examples. This script shows how to launch Access! and do a set of actions:

```
;; This Scripit script will load Access, tell it to
   start dialing
;; then run the Supra ShowModem program and move
   its window to
;; the top left corner of the screen.
;;
;; By: Khalid Aldoseri.
;;
;; Replace this with the full path of where your
   copy of Access is.
      runback ''dh2:Access/Access!_Supra''

;; Wait until all of Access's five windows show up.
;; Note:  If you're not using the chat window
   change the #5 to #4.
      waitfor Access #5

;; Tell Access to Re-Dial. i.e. dial first number.
      menu Phone Re-Dial

;; Run the Supra ShowModem program.
;; Replace this with the full path of  ShowModem.
      runback ''dh2:Access/ShowModem -sAccess''

;; Wait until ShowModem's window shows up.
      waitfor Access Supra Modem
;; Move it to the top right corner of the screen.
      window moveto 640 0
```

The quotes are not important unless you have spaces or commas in the arguments. If you need to specify any CLI command switches when running the program use: runback "programname switches..."

Here's a script that runs WordPerfect, resizes its window to fit half the screen, tells WP to open a new window, resizes that to half as well, and then moves it to the bottom of the screen, giving you WP ready for action with two half screen-sized windows open.

```
;; Run WordPerfect
        runback ''WP:WP''

;; Wait for WP to show up on the Workbench screen
        waitfor Workbench WordPerfect

;; Resize WP's window to 640 by 100
        window resizeto 640,100
        wait

;; Select the menu 'Project' 'New' to open a new
   window.
        menu Project New

;; Wait until the second window shows up.
        waitfor Workbench ''WordPerfect 4.1 -Doc 2''

;; Resize second window to 640,100
        window resizeto 640,100
        wait

;; Move the second window to 0,100.
        window moveto 0,100
```

This is another Word Perfect example. This time, however, it runs WP then loads a specific file.

```
        runback ''WP:WP''
        waitfor Workbench WordPerfect
        menu Project '' Retrieve'' File
        waitfor Workbench
        Retrieve

;; Simulate the user typing ''RAM:Test'' (replace
   this with your own filename)
        key ''RAM:Test''.

;; Click on the Accept gadget
        gadget Accept
```

You can easily merge these last two examples and have Word Perfect load any number of files you like!

As I hope you'll agree, Scripit is a very powerful tool in making our computer easier to use by automating those very repetitive tasks. It becomes even more so when driven by ARexx to control those programs that do not yet support ARexx.

Scripit is a freely distributable, copyrighted program by Khalid Aldoseri. Future enhancements and example scripts can be found in CompuServe's AmigaTech Forum. Support for Scripit is provided in AmigaTech by Khald and Brian Bartlett. Scripit can be found in Lib 12 in AmigaTech Forum.


Free Spirit Software Inc.

**AMI..ALIGNMENT SYSTEM**

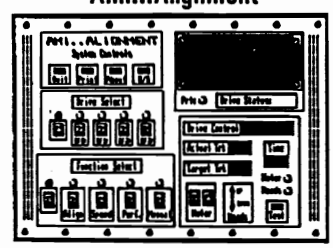
Precision drive evaluation and head alignment system for the Amiga series of computers. The Ami..Alignment System evaluates the mechanical condition of internal and external 3.5" disk drives and provides information for the correction of non-standard conditions.

From a control panel menu, head alignment, drive speed and read/write performance may be tested. Instantaneous readouts are displayed on screen or may be output to printer.

Includes program disk, calibration disk and operations manual. Instructions and diagrams for dissassembly and adjustment of Commodore and third party drives are included.

Diagnose and treat the "flaky floppy" syndrome.

**Ami...Alignment**



See your local dealer or contact:
**Free Spirit Software**
58 Noble St.
Kutztown, PA 19530
215-683-5609
FAX #215-683-8567

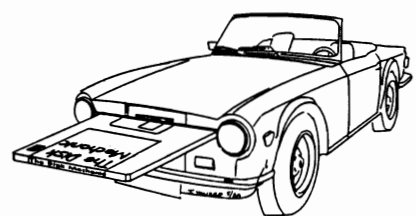
Treat your drives with respect!

**Suggested retail price $49.95**



# The Disk Mechanic

**Give your disks a Tune-up**



**Only $89.95**

The Disk Mechanic is a comprehensive collection of utilities for Amiga DOS, including TuneUp, DoubleBack, DiskRepair, and WorkShop.

**TuneUp:** With TuneUp you can improve hard disk or floppy disk performance by up to 400%. Not a caching program, TuneUp is a sophisticated disk optimizer that actually re-organizes the information on your disk to improve system efficiency. TuneUp is fully compatible with Amiga DOS.

**DoubleBack:** DoubleBack is a high speed archival hard disk back-up program. DoubleBack supports multiple source and destination drives, file selection by name, date, protection, directory, and file note. Back up a full 20 megabytes in under 40 minutes. Selectively copy files between hard disks and partitions. Protect your data!

Sample TuneUp benchmark of some standard 3.5" disks:

| Volume Name | TuneUp | BEFORE | AFTER | IMPROVED |
|---|---|---|---|---|
| Workbench 1.3 | 3:30 | 4.88 | 2.70 | 79% |
| Digi! | 3:45 | 9.44 | 5.22 | 80% |
| WordPerfect | 4:42 | 12.36 | 5.37 | 130% |
| SuperBase | 3:37 | 19.39 | 9.66 | 100% |
| data | 1:43 | 14.30 | 4.68 | 205% |
| archives | 3:40 | 28.43 | 9.52 | 299% |

* Times given are the average of three runs on a standard A2000 internal floppy drive. The TuneUp time is the time required to process the disk (in minutes and seconds). The BEFORE and AFTER times are the time (in seconds) required to get a directory of the disk. The more use a disk has had, the better the improvement!

**Demo disk Available; call for details.**

**DiskRepair:** With DiskRepair it is possible to recover accidently deleted or discarded files, salvage files from damaged disks, and reconstruct a corrupted hard disk. DiskRepair works with hard and floppy disks, and is compatible with the Fast Filing System.

**WorkShop:** For the expert user, the disk WorkShop is a full featured disk file and block editor. Examine and modify disk files, search for viruses, and change menus or requesters in popular programs.

The Disk Mechanic also includes a dozen small CLI based utilities to make managing your disks easier. We are now shipping version 2.5 with full support for Amiga DOS 1.3 and the Fast Filing System. The Disk Mechanic is not copy protected.

Owner inquiries welcome.
Amiga is a trademark of Commodore Amiga Inc.
WordPerfect is a trademark of WordPerfect Corp.
Digi! is a trademark of Aegis Development
SuperBase is a trademark of Precision Software, Ltd.

**Lake Forest Logic Inc.**
28101 E Ballard Road
Lake Forest, IL 60045
(312) 816-6666

MasterCard VISA

# Master The Raster

## *The seduction of the production houses*

**by Derek Grime**

The other night I was enjoying a round of the insomniac's favourite video game, the late night channel hop. Perhaps you've played this game yourself before dawn some morning. Clutching the remote control, you scan through the local television stations, hoping to find something on the tube that is boring enough to put you to sleep or interesting enough to wake you up. After half an hour of the hop, I was wide awake. In fact, I was taking notes. It wasn't the programming that had perked me up, it was the commercials.

I saw a furniture store's prices scroll by in twenty point diamond script. A realtor's houses for sale appeared to have been scanned in with Digi-view. A used car dealer's logo flipped around the screen, an effect that could only have come from Deluxe Paint III.

The Amiga was everywhere.

Last month we covered some of the areas where the Amiga family was facing stiff competition in the desktop video arena. The twin forces of IBM and Apple are gearing up their marketing armies for a major assault on the DTV market.

It has taken them over three years to figure out this latest curve in the marketplace. It will take them a while longer before they figure out just how to approach the desktop video user. They may already be too late. The Amiga has won the war. It now has to hold on to everything that it has gained.

As a desktop video platform the Amiga still has many strengths that the other companies cannot touch. Let's look at some of these abilities that are far beyond mortal computers.

### First and foremost

Acceptance takes a long time. The Amiga is accepted. It didn't happen overnight. There's a lot that can be said about being the new computer on the block. It can be a lonely place. People may not listen to what you're saying. And if they listen, they may not understand.

It took our Amiga awhile to make any friends at all. When it was released, the other companies looked on curiously for a time. In short order, they decided the Amiga was something they could dismiss as a "hackers" computer. It was too wierd to join the gang.

Marketing the Amiga has not been easy. Commodore did as well as it could, but the Amiga has always defied being pigeon-holed. It is a business unit and a games machine. It is a music studio and a video studio. It works with a mouse like a Macintosh. It works with a command line like the IBM. It was a tough sell.

Then something magical happened. No marketing survey could have predicted what happened next. The Amiga did not introduce itself to the public by being carried in through the front door. It came into peoples living rooms first.

Desktop video was the key. Amiga graphics surfaced on "Max Headroom" and a countless number of music videos. The public noticed. The professionals were stunned. It was unheard of that a home computer could be used for network broadcast graphics.

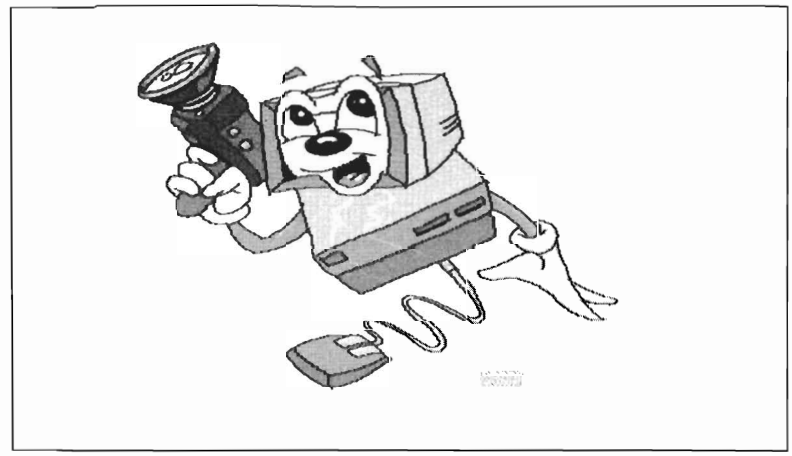
Amiga 1000 CPUs started to appear on television. They were used as props in shows like "Saturday Night Live" and "Miami Vice". Amiga's and Amiga graphics were featured in major motion pictures. Commodore had the hippest computer in Hollywood.

The artists and engineers spread the word amongst themselves. Articles in trade magazines talked about the versatility and power of the new machine. Amiga's became a standard tool in many video studios. They were tinkered with and hacked apart. Amigas were rack-mounted in editing lounges. They were connected to RGB-NTSC video encoders long before consumer genlocks became available. Home made MIDI interfaces and sound samplers got kludged together. The Amiga had found it's niche.

Since it was the first professional video computer it was the first computer many video professionals purchased. For most of these users it will also be the last. They put in the effort once. They rearranged the way they worked to accommodate

the Amiga. Switching to another computer at this late date would be difficult. Other PCs may be interesting, but not worth the energy. Learning a new operating system is no easy chore.

It will be tough for the competition to unseat the Amiga. In the video world it has seniority.



### Master of a thousand disguises

The Amiga is the video chameleon. No machine is more versatile to the video professional. From the smallest wedding video to the largest music video, the Amiga can find a place in every production.

For the hobbyist the Amiga offers great results for a modest cost. It does graphics of every flavour. It animates in two or three dimensions. It digitizes and colourizes. It plays music and talks. It has a male voice and a female voice. It blends, slices and juliennes bit maps.

Despite the power under the hood, it's easy to use. The mouse makes it simple and if you need more control you can learn the DOS. For the small business person who has squeezed the budget to buy an Amiga, profits can be realized very quickly.

For the professional the Amiga can fit anywhere in the organization. It cannot become obsolete because it can fulfill so many functions.

One studio I have consulted with in Toronto purchased their Amiga in the beginning to produce broadcast animation. Two years later, after buying a dedicated workstation costing thousands of dollars the Amiga became the studio's character generator. They purchased a Chyron recently and moved the Amiga to another department where it's being used for scripting and some storyboarding.

While I felt a little disappointed that the Amiga wasn't being used for broadcast anymore, the studio head was thrilled. He explained it to me. Staying up to date costs them more money than you can imagine. Equipment is fresh for such a short time that the resale value on it is almost nothing.

The industry moves that fast. The Amiga was one tool that could handle different jobs equally well. It didn't have to go in the trash like many of their single purpose units do. Like a cat with nine lives, the Amiga has jumped from task to task over the past three years.

Even more lucrative is using the Amiga for many different jobs around your studio. No other machine is as flexible, and that is important in a rapidly changing market. Video and graphics may have been the main reason you bought the Amiga, yet it can do so much more.

Audio production offers some tremendous opportunities. If you feel comfortable with music, why not suggest your clients purchase soundtracks to go with their productions? You may be surprised at the reaction. Most studios have to rely on canned music. There are excellent tracks that can be licensed from commercial musicians. These short songs are used in television commercials and at the beginnings of programs.

For a non-broadcast (that means industrial only) licence, the fee is usually around five hundred dollars. If the client is willing and you have the talent, you can probably arrange a lesser charge that you'll both be happy with.

If it's a case where the music is just an ambient effect behind an announcers voice, a track produced with the Amiga's internal voices may be more than good enough. Produce a portfolio and give it a try. Desktop music could become a successful sideline.

If you are a non-musician you can offer sound effects. With a sampler, a mike and a program like AudioMaster from Aegis, you can create a sonic boom or the squeak of a mouse in minutes. In our case, we had the equipment around gathering dust. When business allowed, we would create sound effects to pass the time. Now sound effects have become part of the business. This is a new addition and the results have been positive. It's fun too, but don't tell your clients that.

There is so much in one box. Some companies moonlight with desktop publishing. Others do accounting or organize databases. Some write articles for the *Transactor for the Amiga*...

The opportunities are as varied as you imagination. No computer does it better... cheaper. Which brings us to the next point.

### Priced too low to mention

There is no other desktop video tool that can deliver half of what the Amiga does for twice the price. Believe it or not, this has it's good points and bad points.

Studios can buy an entire Amiga package with a genlock and software for less than the price of a stand alone titling system. Wise purchasing agents realize that an Amiga can pay for itself even if its only used for the occasional job. The fact that it has the chameleon factor and can balance the books, print out

invoices and play a wicked game of chess are just welcome bonuses. For the smart buyer, it's like getting a computer for free. Not all buyers are smart however...

The fact of the matter is that a few are skeptical of the Amiga's performance because it is so inexpensive. Elitism runs rampant in the video world. They assume that a bargain has to be inferior. It's too good to be true, therefore it must be false.

I'm not suggesting raising the Amiga's price. Heaven forbid. This is where creative selling comes in. You may have heard someone say, "We don't sell computers; we sell solutions".

That's VAR (value added reseller) talk. Some salespeople have had great success with bundled Amiga systems. The A2000 is packaged with a genlock and all the necessary software on a hard disk. It's then sold to studios under a different name.

The price is a lot more than what the individual components would cost. The only custom features of the bundle are the logo on the case and the premium pricetag. I've received countless brochures for incognito Amigas doubling sucessfully as titlers, video editors, slide production units and news graphics systems.

Bundles are not a new idea. In the IBM world, they are commonplace. To get an IBM up to video spec requires some amount of hardware trickery. The bundle includes the extra gadgets, and the bundle is the only way the extra hardware is sold. Amiga owners win again. If we want a genlock, we can purchase it separately. It doesn't come hard-wired into an Amiga 2000.

In fairness, Amiga bundles do have a niche all their own. The corporate buyer is often quite willing to pay extra money to save time. Researching the best hardware and software options is a considerable job. As an extra service, the VAR usually handles all product set-up and support after the sale is made.

How competitive are the Amiga's prices? Here's a real world example. The media services department from a hospital called to enquire what a video graphics system would cost. They needed a genlock, slide output, digitizer, animation and IBM compatibility. The Amiga's price was around $11,000 for all the peripherals and software.

The IBM bundled product that I was competing against had no animation or digitizing features. It was $26,000. Over twice the price! Even scarier, it was not IBM compatible, as the video hardware totally took over the CPU.

Amiga hardware is priced too low to ignore.

### Third party favours

The fourth factor in the Amiga's favour is the wild and wonderful array of third party products that are available only on the Amiga. So much hardware and software has come out that is aimed specifically at creative graphic and video artists. Early hits like Digi-view and the Easyl played an important role in helping the Amiga gain acceptance among professionals.

It's difficult to imagine where the Amiga would be without Digi-view. It's a masterpiece of engineering. Smaller than a deck of cards it scans a black and white image in a red, green and blue pass at twenty-one bit-planes or two million colours for each component. Software converts the information down to the best combination of the Amiga's four thousand and ninety-six colour palette.

The software is doing some incredible work. Diga-view was the only program that could even make use of ham mode graphics for a long time. It showed the world what could be done on the Amiga. Even today, colour scanners for other computers cost thousands of dollars, and paint programs cannot edit the digitized pictures.

Other pioneering hardware was just as ground breaking.

Anakin's Easyl has a pressure sensitive surface. The artist actually felt as though the computer was an extension of the page. There are no wires to get in the way. Because of multi-tasking, the software driver sat in the background waiting to be called.

A-live! offered real time digitizing first. Now that digital frame buffers are used every day some stunning effects are still unique to A-live!

Long before the rest we had affordable genlocks and sound samplers. Some products were Amiga originals, like the Mind Light colour organ.

Software was equally as important in establishing desktop video. Not enough can be said about Deluxe Paint. More than anything else it pulled the Amiga into the creative realm. It remains the number one selling program, far outdistancing the most popular games.

Then came movement. The Amiga could animate rings around anything when it was released. It continues to hold that title today. Aegis' Animator was an important program. So was Deluxe Video. The third dimension appeared with Videoscape and Digi-Paint let us edit HAM for the first time. PageFlipper let us bring HAM to life.

Each of these products contributed to the Amiga's image as a low cost video workstation. Pro's snapped them up first and the consumers were right behind them. Tips were passed back and forth on BBSs. Schematics appeared showing where to cut and solder to improve video output. Many A1000's lost their lives in the pursuit of a better picture.

Public domain software hacks were there at the dawn of the Amiga. Amiga owners swapped anims and graphics like they

were gumcards. Leo Schwab's animations showed what the machine was capable of while poking a fork into some sacred cows. Since 1985, Fred Fish's library has grown to over two hundred disks. There are now hundreds of megabytes of public domain files and the greatest percentage of them are concerned with the visual arts.

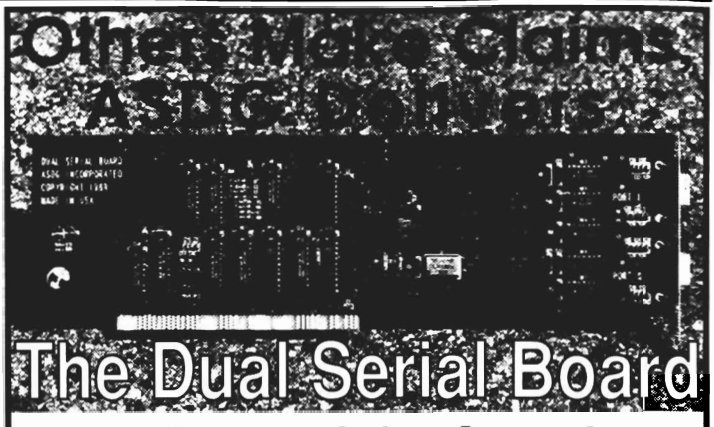
### Tomorrow the world?

Today, the Amiga is established as the number one desktop video computer. It had a head start, remained flexible and affordable, while retaining a extensive base of third party support products.

It's difficult to predict what tomorrow is going to bring. It looks like desktop video could be the major growth area among home computer sales for the nineties. The other companies know this.

For better or worse that means that Commodore has some real competition now. The Amiga is in fine shape today, though colours on-screen and display resolution are becoming issues. These problems are being worked on, and we should see some solid announcements soon.

Where does that leave the creative artist who has invested time and money on the Amiga? In good shape. Your taste is superb. The Amiga still remains the best desktop video system available in the world today.


**Fast, Compatible, Complete!**

**● Two high speed RS-232C serial ports for the Amiga 2000. ● IBM PC AT style (full 7 wire plus RI and DCD) connector for each port. ● Send and Receive at 110 to 115200 baud. ● Complete Amiga system software (both Exec *and* DOS). ● Using our exclusive Serial DisPatcher, *any* program which uses "serial.device" can use any port *without* patches. ● Public domain and shareware software on the distribution diskette. ● One year warranty.**

**ASK YOUR DEALER!**

**ASDG Incorporated**
925 Stewart Street
Madison, WI. 53713
(608) 273 - 6585

Amiga is a trademark of Commodore-Amiga, Inc. IBM PC AT is a trademark of IBM Corp.

# ☆ ☆ ☆ ReSource ☆ ☆ ☆

## Intelligent Interactive Disassembler for Amiga Programmers

*"**ReSource** makes full use of the Amiga windowing environment ... One of the features that distinguish **ReSource** is its ability to produce output which is compatible with most popular assemblers ... a bargain in this world of overpriced software." — Andre Marino*

**ReSource** is an interactive disassembler for the Amiga computer. Written entirely in assembly language, this is the most powerful disassembler available on *any* personal computer! Over 700 menu functions allow you to:

- ✔ Load from *any* file, read tracks, or disassemble memory directly.
- ✔ Save ALL, allowing you to continue disassembling later.
- ✔ Save to a ".asm" file, partial saves supported.
- ✔ Automated symbol creation:
  JSR -$1E(A6) becomes JSR _LVOOpen(A6)
  MOVE.L #$3EE,D0 becomes MOVE.L #MODE_NEWFILE,D0
  Over 5000 symbols supported!
- ✔ Incredibly smooth scrolling.
- ✔ Full key rebinding supported. Load/save keytables.
- ✔ Up to 38 user-defined macros available. Load/save/overlay macros, looping/conditionals supported, variable execution speed.
- ✔ Numeric and text processing functions allow you to automate many things that used to make disassembling so slow!
- ✔ Search function uses full ARP wildcards.
  Search forwards, backwards, or use the special bi-directional search!
- ✔ The *fastest* text rendering available on the Amiga.
- ✔ Forward and backward referencing allows you to quickly examine referenced code/data, and return to your original position.
  Nested references supported.
- ✔ Automatically/manually create labels. Manually create/edit/delete labels, symbols, end-of-line comments, full-line comments.
- ✔ Special support for "C" programs, including "A4" register data referencing.

**US$75** **If you're serious about disassembling code, look no further!** **A$90**

To order **ReSource V3.05**
Send check or money order
Credit cards accepted

The Puzzle Factory
P.O. Box 986
Veneta, OR 97487
U.S.A.
(503) 935-3709

Glen McDiarmid
28 Marginson St.
Ipswich, QLD
AUSTRALIA 4305
(07) 812-2963

# Assembly Language Programming

## *Object Files and Linking (Part 2)*

**by Jim Butterfield**
*Copyright © 1989 Jim Butterfield*

*Jim Butterfield is a Toronto Programmer, author, lecturer whose many articles, books, lectures and television programs have made his name a household word among Commodore users throughout the world since the early days of the KIM-1 microcomputer.*

We'll continue with the business of building object files separately and combining them with the linker. This time, we'll develop more complex systems, and will look briefly at alternatives such as *include files* and *macros*.

As a related subject, we will take a closer look at subroutines, and how information may be passed back and forth between a main program and the subroutine it calls. In this area, we'll touch upon the C language and its most common subroutine communications method.

Last time, we wrote two programs separately, generated two *object* files, and then had the linker put them together into a single executable program. We're going to follow this trail a little further, building a small *archive format* file containing *scanned library* subroutines that the linker can call upon as it needs. Then we'll sit back for a moment and look at how this technique may be used to help us construct very large assembly language programs.

If you kept program *RegDumper.asm* from last time around, good. We'll use it again. If not, you'll need to type it in from Listing 2. If you assembled this program and kept the object program, *RegDumper.o*, that will be useful. Otherwise, assemble the source (to object only) so that you'll be ready to continue with the new project.

Similarly, it will be handy if you have kept the AmigaBasic program *HunkViewer*. This program will allow you to view the files we create, and see how they work together. If you don't have this program, load up AmigaBasic and type in the program from Listing 1.

### Looking back

In the previous session we created a main program, *Hello1*, and a separate subroutine, *RegDumper*. We planned that program *Hello1* would call *RegDumper* as it ran. So: we put together these two separately-assembled modules, asking the linker to fit them into a single program for us.

We might have done this with the command:

```
LINK Hello1.o RegDumper.o TO Hello1
```

This signals that our program will be made up of both *Hello1.o* and *RegDumper.o*, blended together as an executable file called *Hello1*. But we used another way to do the job:

```
LINK Hello1.o TO Hello1 LI RegDumper.o
```

This signals that our program will be made from *Hello1.o*; but if there are any unmatched XREF items, the linker will search (library) *RegDumper.o* for them. If a matching XDEF is found, *RegDumper.o* will be included as part of executable file *Hello1*.

This second method is the one that interests us. In this case, the linker will include either the whole of *RegDumper.o* or none of it. That's because *RegDumper.o* is a single *program unit*.

When the linker searches a library item and finds something that it needs, it takes the entire *program unit*, which will consist of one or more hunks. In this session, we're going to create an *archive format* file that consists of more than one program unit. The linker will scan these units within the library file. When it finds an item it needs, it includes the program unit concerned, not the whole file.

Let's build an archive format ("scanned library") file. We already have one unit, *RegDumper.o*. We will write another subroutine, and combine it with *RegDumper*. When the two are lumped together as an archive format file, we'll write a simple program that will cause the linker to pick out only the program unit that is needed.

### A second subroutine

Some time ago, we wrote a piece of code that would convert a 32-bit (unsigned) binary number into a string of ASCII digits,

ready to be printed. That seems like a good candidate for a subroutine to be included in our library.

Call the following file *LongToAscii.asm* when you enter it with your favorite editor. We'll start by giving an identity to the program unit and to the hunk, and supplying the external definition that other programs will use to call the subroutine.

```
IDNT BinAscii
SECTION ConvAsc,CODE
XDEF MakeAscii
```

We should comment this program nicely. In the future, we might want to look here to remember what we are doing.

Most Amiga system calls allow registers D0, D1, A0 and A1 to be changed, carefully preserving the other registers. We'll follow this rule when we write the subroutine; it will make things easier to remember.

The calling routine needs to set up some things before it goes to the subroutine. A memory area must be set aside to hold the ASCII characters that the subroutine will calculate. A pointer to this memory space should be placed into register A2, and its size in bytes into register D2. The longword numeric value to be converted will be placed into register D0. All this is noted in the comments.

```
; Subroutine:  Convert unsigned longword to ASCII
;       string
;       -value received in D0;
;       -result placed in string at (A2)
; Calling program MUST provide:
;       longword value to be converted in D0;
;       a buffer to hold result string;
;       address of this buffer in register A2;
;       size of this buffer in D2;
; Results:
;       Ascii string, zero-suppressed, in buffer;
;       Register D0 will contain zero if buffer
;       is sufficiently large; non-zero indicates
;       overflow.
; Registers:
;       Standard;  Registers D0, D1, will be changed;
;       all others will be preserved.
;
```

As we enter the subroutine, the buffer area will be blanked out to SPACE characters.

```
; First, clear string area to SPACEs.

MakeAscii   move.l  d3,-(a7)    ; save D3 contents
            moveq   #$20,d1     ; space character
            move.w  d2,d3       ; size of output area
            subq.w  #1,d3       ; .. minus 1 for DF
Clear       move.b  d1,0(a2,d3.w)  ; store a SPACE
            dbf     d3,Clear       ; loop to do more
```

We will be calculating digits from the right-hand side of the number. Register D3 will be an index to the output digit that we are working on; it starts high and goes down.

```
            move.w  d2,d3   ; number of bytes
            subq.w  #1,d3   ; ..minus one for looping
```

We saw this calculation algorithm a couple of sessions ago. It uses the slow but elegant DIVU (divide unsigned) command. Since the 68000 DIVU command produces a quotient that is only 16 bits in size, we need to divide in two chunks: the high word first, and then the low. Here comes the high-end division.

```
; Calculate digits from low end of number.

DigLoop     swap    d0      ; high/low word swap
            moveq   #0,d1
            move.w  d0,d1   ; high word to D1
            beq.s   Skipper ; zero, don't divide
            divu    #10,d1  ; divide hi word
```

Now the division for the low end of the number.

```
Skipper     move.w  d1,d0   ; new hi word value
            swap    d0      ; restore hi/low order
            move.w  d0,d1   ; set low value
            divu    #10,d1  ; continue division
```

The division is complete. We re-form the quotient into register D0, and then swap the remainder into the lower part of D1, where we can change it to ASCII. Since we are dividing by ten, the remainder must be in the range of zero to nine.

```
            move.w  d1,d0   ; new lo word value
            swap    d1      ; get remainder
            or.b    #$30,d1 ; make it Ascii
            move.b  d1,0(a2,d3.w)  ; put at low end
```

If our number has been reduced to zero by division, we won't need to generate any more digits. This method is speedy with small numbers, since it will stop whenever the digits are complete.

```
            tst.l   d0          ; any more digits?
            dbeq    d3,DigLoop  ; yes, go back
```

This illustrates the elegant two-way test of the DBCC instructions. If D0 is zero (no more digits needed), the DBEQ instruction does nothing, allowing the processor to "fall out" of the loop. Otherwise, D3 is decremented. If D3 goes below zero (no space left in buffer), the loop is not taken. Otherwise (more digits and space still available), the program loops to do more work.

```
            move.l  (a7)+,d3    ; restore D3
            rts
            end
```

When you have entered the above program, save it to disk and then assemble it to an *object* file named *LongToAscii.o*. Do *not* link this file; it won't work as an executable program.

At this point, you should have two object files: *RegDumper.o* and *LongToAscii.o*, each one a program unit. We'll put them together into a scanned library by simply joining them:

```
JOIN RegDumper.o LongToAscii.o AS Combo.lib
```

The result, file *Combo.lib*, is a scanned library file (sometimes called an *archive format* file) containing two program units. You could of course create library files containing dozens of program units.

Use the AmigaBasic program *HunkViewer* to look at the various files. You'll see that *Combo.lib* contains two units. In this case, each program unit contains one hunk only, but multi-hunk units would be quite OK.

### Simple program: *HelloNum*

Let's write a simple program to use our newly-formed library file. Again, it will be a variation of the good old "Hello World" code. This time, we'll arrange to print a numeric value before our message. Call the following source file *HelloNum.asm*. It starts out with a few comments and our system call definitions.

```
; Hello program
; Startup code, CLI only
; Equates for Exec Library
            _LVOOpenLibrary   equ  -$228
            _LVOCloseLibrary  equ  -$19E

; Equates for Dos library
            _LVOOutput        equ  -$3C
            _LVOWrite         equ  -$30
```

One more definition: the label *MakeAscii* will be found by the linker, which will also bring in any necessary code. The 'usual' CLI startup code follows.

```
   XREF MakeAscii

; Label MakeAscii must be found during "link"

            MOVE.L  $4,A6          ; Exec base
            LEA     DosName(PC),A1 ; Dos name string
                                   ; pointer
            MOVEQ   #0,D0          ; any version
            JSR     _LVOOpenLibrary(A6)
            MOVE.L  D0,A6          ; Dos base pointer
            BEQ.S   Exit           ; Zero, quit
            JSR  Main              ; Do the job
            MOVE.L  A6,A1          ; Dos base pointer
            MOVE.L  $4,A6          ; Exec base pointer
            JSR     _LVOCloseLibrary(A6)
Exit:       RTS
```

Here comes the program's main job. We will get an output handle. Then we must prepare for number output. We will need a buffer; let's take it from the stack with **LINK**. The buffer address is moved to A2, and its length (hex 20 or decimal 32) to D2. That's a much bigger buffer than we need, of course, but our library routine can handle it.

```
Main:
; Main Job
;
            JSR     _LVOOutput(A6) ; output handle
            move.l  d0,d7   ; save handle in D7
            link    A5,#-$20       ; gimme 32 bytes
            lea     -$20(A5),A2    ; put address in A2
            moveq   #$20,D2        ; and length in D2
```

The buffer information is in place. Pick a number – decimal 12345 suits me fine – place it in register D0, and call routine *MakeAscii*. When the assembler looks at this program, it won't have the foggiest idea where *MakeAscii* will be, or what it will contain; but it will 'trust' the linker to sort things out.

```
            move.l  #12345,D0 ; pick a number
            jsr     MakeAscii ; make ASCII decimal
```

When *MakeAscii* returns, A2 and D2 will still contain their original values. Now to set things up for the Write operation: a little register-shuffling is appropriate.

```
            move.l  A2,D2      ; address of buffer
            moveq   #$20,D3    ; size of buffer
            move.l  d7,d1      ; output handle
            jsr     _LVOWrite(A6)  ; print it!
            unlk    A5         ; return buffer space
```

Finish by sending the "Hello, World!" message. Since we did not arrange to print a NewLine character after the number, the "Hello" will appear on the same line.

```
            move.l  d7,d1   ; Handle
            LEA     Message(PC),A0 ; Address of
                                   ; Message
            MOVE.L  A0,D2   ; ...to D2
            MOVEQ   #14,D3  ; Length of Message
            JSR     _LVOWrite(A6)  ; Send message
            RTS

Message     dc.b    'Hello, World!',$0a
DosName     dc.b    'dos.library',0
end
```

Save the text, and assemble it as object file *HelloNum.o*. We're ready to fold our pre-assembled subroutine into the result. Command **BLINK HelloNum.o TO HelloNum LI Combo.lib** and the magic will take place. Even though library file *Combo.lib* contains code for both *RegDumper* and *MakeAscii*, only the *MakeAscii* unit (one hunk, in this case) is added to your program. *RegDumper* is ignored, since it's not needed.

## Taking stock

Why would we compile program modules in advance, and ask the linker to put them together? There are several reasons.

First, it's a good way to build a sound library of pretested subroutines. You won't need to worry about writing a certain task, such as binary-to-ASCII conversion, over and over.

Next, your work will be better organized. The code you are writing won't get lost in a clutter of detail if you segregate various jobs into library elements. And if you happen to use the same label internally within several of these modules – labels such as LOOP and EXIT are popular – they won't clash.

Can your assembler cope with a really huge program? Sooner or later, all assemblers can run out of memory, even with the Amiga's huge RAM resources. Breaking up your program guards against the chance of your assembler ever failing in this way.

Large programs assemble slowly. Small programs assemble fast. It's more than proportional, since doubling the size of a program will usually cause it to need well over twice the assembly time. Pre-assembling smaller units can result in a net savings of time. This is especially true, of course, if some parts need several assemblies as bugs are worked out.

## Includes and macros

Some of the above arguments could also be made on behalf of two other methods of 'modularizing' programs. *Include* files contain 'canned' code that may be brought into your source program. This gives you pre-tested text, and saves you typing. *Macros* (which often come from include files) allow you to invoke a piece of code at any point within your program. We won't go into great detail on them here.

The include files you are likely to meet in your Amiga system will contain mostly definitions of labels. For example: if your program has obtained a FileInfoBlock and placed its address into register A4, the file size may be obtained with a command such as **MOVE.L** fib_Size,D0...allowing an include file (in this case, *dos.i*) to supply the (offset) value for fib_Size.

You can put code into include files, of course. In that case, your source program would command **INCLUDE** to add extra instructions to your program.

Macro definitions are often located within include files; they can be flexible and useful. If you decide that you'd like to type **CALLEXEC OpenLibrary** and have it expanded to the two instructions **MOVE** 4,A6 and **JSR** _LVOOpenLibrary(A6), a macro will do the job quite nicely for you. Macros are more flexible than straight include or library files, since 'intelligent' macros can adapt themselves according to whatever way they are called.

Both include files and macros make your source program bigger, and make your assembler work harder. Sometimes they are best; at other times the linked library approach is superior.

## Multiple libraries

The linker may be asked to search more than one library file. For example, if we had defined our system calls with **XREF** statements rather than **EQU**, the linker would need to look for those values; and it wouldn't find them in *Combo.lib*. No problem: the following command would do the job.

```
BLINK HelloNum.o TO HelloNum LI Combo.lib Amiga.lib
```

There are two libraries to be searched in this case. Libraries are searched in the order they are given. If the same desired **XDEF** appears in both files, the first one found would be used. For our program, there would be no conflict, and the library files could be typed in any order.

## Subroutine call methods

You often want to send information ('parameters') to a subroutine, and usually expect to receive information back. There are several ways you may pass parameters to and from a subroutine.

The simplest, most obvious, and most popular method is to put the information into one or more registers. For example, when we called subroutine *MakeAscii* we placed the number to be processed into register D0. Sometimes there's too much information to fit into a few registers. In that case, we can put the data into an area memory; a register will be set up with a pointer to this information. Again, *MakeAscii* used this method when it received a pointer to a buffer area; it stored the data it generated into that area.

Sometimes registers are at a premium, and the register passing method seems less convenient. This is especially true when one subroutine calls another, which calls another, and so on. You might run out of registers for the various calls; or, more likely, the various registers needed to pass parameters would be in conflict among the several subroutines. You could always carefully save the registers that needed to be preserved. Or: you could put the parameters on the stack.

## Putting parameters on the stack

As an example, suppose I have a pointer and a longword value, and I want to pass these to a subroutine called CALC. Suppose, further, that I don't want to use registers to pass the information. I could push the two values to the stack and make the call; after the subroutine returns, I must remember to remove the values from the stack. Let's do this with several methods.

```
Method1     MOVE.L  Pointer,-(A7); pointer to stack
            MOVE.L  Value,-(A7)  ; 4-byte value to
                                   stack
            JSR     CALC
            ADDQ.L  #8,A7   ; clean 8 bytes from stack
            ...

Method2     MOVE.L  Pointer,-(A7)
            MOVE.L  Value,-(A7)
```

```
        JSR     ALC
        LEA     8(A7),A7
```

The two above methods clean the stack by adding eight, which restores the stack pointer to its previous value. There is no important difference between the two, except that **ADDQ** cannot be used for values greater than eight. Variables *Pointer* and *Value* might be reference in other ways, of course; they would often already be in registers. **JSR** might be replaced by **BSR** if CALC is in the same program hunk.

Other programmers like to set up a stack frame to hold the values, with the principle in mind that it's somewhat easier to clean up when the job is done. The following code demonstrates the method.

```
Method3 LINK    A5,#-8
        MOVE.L  Pointer,-4(A5)  ; pointer to stack
        MOVE.L  Value,-8(A5)    ; 4-byte value on
                                  stack
        JSR     CALC
        UNLK    A5    ; clean 8 bytes from stack
```

This uses extra time and code, but you may feel that the extra neatness is worth it. Finally, if your data is in registers to begin with (say, Pointer is in A2 and Value is in D3), another method is sometimes used.

```
Method4 MOVEM.L A2/D3,-(A7) ;  params to stack
        JSR     CALC
        MOVEM.L (A7)+,A2/D3  ; restore params and
                               stack
```

This last method also restores the registers, a feature which often may not be needed. Be careful with this one: the registers may not go on the stack in the desired order. Keep in mind that all Address registers go to the stack first, followed by Data registers; within each group, the higher numbered registers are stored first. In this case, they go on in the desired order (Pointer, A2, first).

## The stack and C calls

Several languages, including C, typically use the stack to pass parameters to subroutines; they often expect a return value, if any, in D0. Let's look at a simple function call in C.

```
...
char *name, *game, *p;
...
p=match(name,game);
...
```

The upper line says, "*name* is the name of a pointer to a string, and so is *game* and *p*". The lower line says, "call subroutine *match*, supplying it with pointers *name* and *game*"; we expect to get back a pointer to a string, which we will assign to variable *p*. How does the C compiler set up this subroutine call?

Before the call, it moves the two parameters to the stack. Pointers *name* and *game* will be moved to the stack with instructions like **MOVE.L** game,-(A7)... **MOVE.L** name,-(A7). Note that the second parameter goes to the stack first. Then there comes the subroutine call to *match*; upon return, the stack will be cleaned with an instruction similar to **LEA** 8(A7),A7. At this point, the program will expect to find a pointer in D0 which may be taken as the value of variable *p*.

What will happen within the subroutine? The stack will be 'peeked' rather than 'pulled'. The subroutine will know that A7 points to the four-byte return address. Pointer *game* will be found at 4(A7), and pointer *name* at 8(A7). It would not be unusual to see code such as **MOVE** 4(A7),A0... **MOVE** 8(A7),A1 which brings the parameters into registers where they can be used. The stack itself will be left as is; the calling program will clean it up after the return.

We are studying assembly language rather than C; does the above matter to us? It might. We could use this information to help us debug a C program. More importantly: armed with this kind of data, we can opt to write subroutine *match* in assembly language and produce an object file. Following this, we could write a C program that called *match*, and the linker would fit the two together. C convenience and machine language speed? It's not hard. Keep in mind, however, that C can generate good code on its own; you'll need to have good reasons to try for that extra speed edge.

## Looking at *Amiga.lib*

In past programs, it was suggested that you ask the linker to dig out the _LVO labels for your program by looking in a library file called *amiga.lib*. This file comes as part of the Commodore developer's disk. You may have assumed that all the file contains is system labels. There's more; and we can get a better glimpse of the workings of libraries and the C language by looking more deeply into this file.

You might like to use AmigaBasic program *HunkViewer* to look at the *amiga.lib file*. The following describes part of file *amiga.lib*; your version may not be exactly the same.

When we programmed our Assembly Language exercises, we used **XREF** for labels such as _LVOWrite. The assembler left the value of _LVOWrite undefined, with a note in the object file asking the linker to look for it. The **BLINK** command would contain the phrase **LI** Amiga.lib, and would start to search the file.

Amiga.lib contains a lot of program units – hundreds of them. The linker would keep looking; about hunk 109 it would find the equivalent of the assembler statement:

```
_LVOWrite EQU $FFFFFFD0
```

The above $FFFFFFD0 value is the 'twos complement' for -$30, the correct offset for _LVOWrite. The linker would put the value it has found into the proper places in your program, and

your JSR ???(A6) would become JSR -$30(A6); when the program runs, this will call a subroutine at the proper offset within the DOS library.

The linker, in principle, does one more thing. When it finds the label, it will take the programming code from that program unit, and add it to your program. In this case, however, there isn't any code; just labels. Hunk 109 and its associated units is empty, and contains only external label definitions (called 'ext_abs', external absolute).

In order to use _LVOWrite correctly we must set up register A6 with the register base (in this case, the DOS library base), and put appropriate values ("arguments") in D1, D2, and D3. Not hard for us to do in assembler, but other languages may not be happy with this register-arranging.

So: a language such as C might like to call the Amiga library in a gentler way. Here's a command in C that wants to call exactly the same function:

```
returnedlength = Write(file,buffer,length);
```

There is no _LVO prefix to the Write command. Let's trace this call as it invokes Amiga.lib. Variables *length* (an integer value), *buffer* (an address), and *file* (a file handle, four bytes) are pushed to the stack. Now C will set up an XREF subroutine call to label *_Write*; note that an underscore character has been added to the label. When the subroutine returns, the C program will clean up the stack and put away the result. Label _Write is left for the linker to find.

So the C object program is linked with Link, and Amiga.lib is searched. About hunk 81, the linker finds an XDEF, external definition, for _Open. It's attached to the following code:

```
XDEF _Write     ; here it is!
XREF _DOSBase   ; but I need this!
_Write      MOVEM.L   D2-D3/A6,-(A7)  ; save regs
            MOVEA.L   _DOSBase,A6     ; set up
                                      ; library base
            MOVEM.L   $10(A7),D1-D3   ; set up param
                                      ; registers
            JSR       -$30(A6)      ; call _LVOWrite
            MOVEM.L   (A7)+,D2-D3/A6 ; restore regs
            RTS
```

This kind of code is often called a 'stub routine'. There is real code in this program unit; it will be tacked onto the program. The code shown above is assembler source; the actual code within Amiga.lib is machine language (it's been assembled), but you'll find the above more readable.

The XDEF says, "if you're looking for _Write, here it is". The following XREF says, "...but you must tell me where you have stored the value of the DOS library base". Label _DOSBase will be defined within the main program, usually by the startup code.

The first and last MOVEM commands save registers to the stack so that they will not be disturbed. Note that register D1 will be affected by the program, but under Amiga's register conventions, we don't need to protect registers numbered 0 or 1. The value of _DOSBase, established somewhere in the main code, will be placed into A6.

MOVEM.L $10(A7),D1-D3 is interesting. It's the first time we have seen MOVEM (Move Multiple registers) used in this way. Instead of directly changing the stack by pushing or pulling values, we 'peek' them out. We do not use -(A7) or (A7)+, so the stack pointer is not changed. We reach out from the current stack pointer: past the three registers we have saved (twelve bytes), and past our subroutine return address (four more bytes). Above those sixteen (hex 10) bytes, we'll find the three parameter values pushed to the stack by the calling program.

MOVEM will reach the last-stored values first, as it 'walks up' the stack area. It fills the lower numbered registers first. So it copies *file* into D1, *buffer* into D2, and *length* into D3. With all registers set up, the subroutine now calls the value corresponding to _LVOWrite.

In seeing this whole sequence, we get a better view of how the various languages work. We also get a chance to interface Assembler code with other language programs. If you wanted, you could call _Write or any similar function directly from Assembler. If you do so, remember that you must first define _DOSBase. Alternatively, you could create your own version of _Write, put it into an archive file, and have your C programs use this custom version instead of the "standard".

If you do not have the Amiga.lib file, you can get _LVO definitions by using a freely distributable substitute, *SmallLib*. SmallLib contains only label definitions; no code.

If you disassemble a program written in a language such as C that loves to pass parameters on the stack, and loves to nest subroutines, you'll see a lot of stack work. A main program might push a whole series of values to the stack before calling a subroutine...which peeks them out and puts them into registers. That subroutine decides to call another one, and back go the values to the stack. The new call pulls them off, and then decides that it needs to call yet another subroutine and...there they go again!

You may find the same value pushed to the stack over and over again. It seems inefficient, but it helps to keep the data flow distinct and unambiguous. You, as the program reader, may be making endless notes as to what has been stacked where. You may even utter a solemn oath that you will never write code such as this. In assembly language, you will seldom need to do so.

### Listing 1: *HunkView*

```
PRINT "Simple Object/Loadable Analyzer .. Jim Butterfield"
DATA "CODE","DATA","BSS"
FOR j=1 TO 3
```

```
 READ HunkType$(j)
NEXT j
DATA "ext_symb","ext_def","ext_abs","ext_res"
FOR j=0 TO 3
 READ SymType$(j)
NEXT j
DATA "ext_ref32","ext_common","ext_ref16","ext_ref8"
FOR j=1 TO 4
 READ RefType$(j)
NEXT j
BlokSize%=2500
LINE INPUT "File to analyze? ";Fil$
OPEN Fil$ FOR INPUT AS #1
 RamType%=CVI(CHR$(0)+INPUT$(1,1))
 Magic&=CVL(CHR$(0)+INPUT$(3,1))
 IF Magic&=1011 THEN
  PRINT "An Executable File!"
  NameSize&=CVL(INPUT$(4,1))
  WHILE NameSize&>0 THEN
   HunkData$=INPUT$(4*NameSize&,1)
   GOSUB NameTrim
   PRINT CHR$(34);HunkData$;CHR$(34)
   NameSize&=CVL(INPUT$(4,1))
  WEND
  Hunks&=CVL(INPUT$(4,1))
  PRINT "It has";Hunks&;"hunks"
  HunkData$=INPUT$(4*Hunks&+8,1)
 END IF
 IF Magic&>998 AND Magic&<1015 THEN
  IF Hunks&=0 THEN PRINT "Looks like an OBJECT file..."
  Hunks&=0
  Live%=0
  WHILE Magic&>998 AND Magic&<1015
   IF Magic&=1010 THEN
    PRINT:Live%=0
   ELSE
    IF Live%=0 THEN Live%=1:PRINT "Hunk";Hunks&;:Hunks&=Hunks&+1
   END IF
   IF Magic&<1004 OR Magic&=1009 THEN
    Size&=CVL(INPUT$(4,1))
    Style%=Magic&-1000:IF Style%>3 OR Style%<0 THEN Style%=0
    IF Style%>0 THEN
     PRINT " (";HunkType$(Style%);":";
     IF Size&=0 THEN
      PRINT " Empty)";
     ELSE
      PRINT Size&*4;"bytes)";
     END IF
     IF Magic&=1003 THEN Size&=0
    END IF
    IF Magic&=999 THEN PRINT " UNIT";
    IF Magic&=1000 THEN PRINT " SECTION";
    IF Size&<>0 THEN
     GOSUB DataRead
     IF Magic&=999 OR Magic&=1000 THEN
       GOSUB NameTrim
       PRINT ":";CHR$(34);HunkData$;CHR$(34);" ";
     END IF
    END IF
   END IF
   IF Magic&<=1006 AND Magic&>=1004 THEN
    Size&=CVL(INPUT$(4,1))
    WHILE Size&>0
     Hunk&=CVL(INPUT$(4,1))
     GOSUB DataRead
     Size&=CVL(INPUT$(4,1))
    WEND
   END IF
   IF Magic&=1007 OR Magic&=1008 THEN
    Type%=CVI(CHR$(0)+INPUT$(1,1))
    SymLen&=CVL(CHR$(0)+INPUT$(3,1))
    WHILE SymLen&+Type%<>0
     PRINT
     PRINT "  ";
     IF Type%<4 THEN
      PRINT SymType$(Type%);":";
     ELSEIF Type%>128 AND Type%<133 THEN
      PRINT RefType$(Type%-128);":";
     ELSE
      PRINT "???";":";
     END IF
     HunkData$=INPUT$(4*SymLen&,1)
     GOSUB NameTrim
     PRINT TAB(15);HunkData$;
     IF Type%=130 THEN
      CommSize&=CVL(INPUT$(4,1))
     END IF
     IF Type%>=129 AND Type%<=131 THEN
      RefSize&=CVL(INPUT$(4,1))
      RData$=INPUT$(4*RefSize&,1)
     ELSE
      SymVal&=CVL(INPUT$(4,1))
     END IF
     Type%=CVI(CHR$(0)+INPUT$(1,1))
     SymLen&=CVL(CHR$(0)+INPUT$(3,1))
    WEND
   END IF
    IF EOF(1) THEN
     Magic&=0
    ELSE
     RamType%=CVI(CHR$(0)+INPUT$(1,1))
     Magic&=CVL(CHR$(0)+INPUT$(3,1))
    END IF
  WEND
 ELSE
  PRINT "Not an Object/Executable file."
 END IF
CLOSE 1
PRINT
PRINT Hunks&;"hunks in this file."
END
DataRead:
 Blox%=INT(Size&/BlokSize%)
 Tail%=Size&-Blox%*BlokSize%
 BSize%=BlokSize%
 FOR k=0 TO Blox%
  IF k=Blox% THEN BSize%=Tail%
  HunkData$=INPUT$(4*BSize%,1)
 NEXT k
RETURN
NameTrim:
       WHILE RIGHT$(HunkData$,1)=CHR$(0)
        HunkData$=LEFT$(HunkData$,LEN(HunkData$)-1)
       WEND
RETURN
patch:
OPEN "ram:crapper" FOR OUTPUT AS #2
PRINT#2,MKL$(1004);MKL$(4521);
WHILE NOT EOF(1)
 a$=INPUT$(4,1)
 PRINT#2,a$;
WEND
CLOSE 2
CLOSE 1
```

## Listing 2:*RegDumper.Asm*

```
IDNT  FirstPart
SECTION DumpRegisters,CODE

; hex register dump     Jim Butterfield
; input needed:  none
; output supplied:  hex dump to standard output
; no registers affected

   XDEF RegDump

; Exec library calls
   _LVOOpenLibrary   EQU -$228
   _LVOCloseLibrary  EQU -$19E

; Dos library calls
   _LVOOutput        EQU -$3C
   _LVOWrite         EQU -$30
```

```
RegDump:

   MOVEM.L d0-d7/a0-a7,-(A7)
   MOVE.L  4,A6
   LEA     DosName(pc),A1
   MOVEQ   #0,D0
   JSR     _LVOOpenLibrary(A6)
   TST.L   D0
   BEQ.S   ForgetIt
   MOVE.L  D0,A6
   JSR     _LVOOutput(A6)
   MOVE.L  D0,D7
   LINK    A5,#-$24
   LEA     -$24(A5),A4
   LEA     OutLine(pc),A3
   MOVEQ   #OutLen-1,D0

CopyStr
   MOVE.B  (A3)+,(A4)+
   DBF     D0,CopyStr        ;word size only

; ready to run through registers
; D2 and D3 will not be disturbed during loop

   LEA     -$24(A5),A4
   MOVE.L  A4,D2             ;address of string
   MOVEQ   #OutLen,D3        ;length of string
;
   MOVEQ   #0,D4             ;start with reg 0

RegLoop:
   MOVE.B D4,D5
   OR.B    #$30,D5
   MOVE.B  D5,1(A4)
   MOVE.B  D5,13(A4)
   MOVE.W  D4,D5
   ASL.W   #2,D5
   MOVE.L  4(A5,D5.W),D0
   LEA     11(A4),A0
   BSR.S   ConvertHex
   MOVE.L  36(A5,D5.W),D0
   LEA     23(A4),A0
   BSR.S   ConvertHex
   MOVE.L  D7,D1             ;outhandle
   JSR     _LVOWrite(A6)
   ADDQ.W  #1,D4             ; next register
   CMP.W   #8,D4             ; all registers done?
   BLT.S   RegLoop

   UNLK    A5
   MOVE.L  A6,A1
   MOVE.L  4,A6
   JSR     _LVOCloseLibrary(A6)

ForgetIt
   MOVEM.L (A7)+,d0-d7/a0-a7
   RTS

ConvertHex:
   MOVEQ   #7,D6
   MOVEQ   #0,D1

HexLoop
   MOVE.B  D0,D1
   LSR.L   #4,D0
   AND.B   #$F,D1
   OR.B    #$30,D1
   CMP.B   #$39,D1
   BLE.S   Numeric
   ADDQ.B  #7,D1

Numeric
   MOVE.B D1,-(A0)
   DBF     D6,HexLoop
   RTS

DosName dc.b 'dos.library',0

OutLine dc.b 'Dx=........ Ax=........',$A
OutLen  EQU  *-OutLine

   end
```

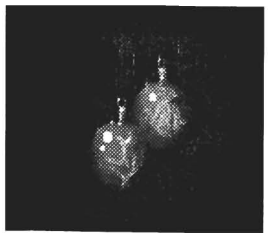
## Listing 3: *HelloNum.asm*

```
; Hello program
; Startup sequence, CLI only
; Equates for Exec Library

_LVOOpenLibrary    equ   -$228
_LVOCloseLibrary   equ   -$19E

   XREF   MakeAscii

; Label RegDump must be found during "link"

   MOVE.L $4,A6              ; Exec base
   LEA    DosName(PC),A1     ; Dos name string pointer
   MOVEQ  #0,D0              ; any version
   JSR    _LVOOpenLibrary(A6)
   MOVE.L D0,A6              ; Dos base pointer
   BEQ.S  Exit               ; Zero, quit
   JSR    Main               ; Do the job
   MOVE.L A6,A1              ; Dos base pointer
   MOVE.L $4,A6              ; Exec base pointer
   JSR    _LVOCloseLibrary(A6)

Exit:

   RTS

DosName   dc.b  'dos.library',0

Main:
; Main Job
; Equates for Dos library

_LVOOutput         equ   -$3C
_LVOWrite          equ   -$30

;
   JSR    _LVOOutput(A6)     ; Get output handle
   move.l d0,d7              ; save handle in D7
   link   A5,#-$20
   lea    -$20(A5),A2
   moveq  #$20,D2
   move.l #12345,D0
   jsr    MakeAscii
   move.l A2,D2
   moveq  #$20,D3
   move.l d7,d1
   jsr    _LVOWrite(A6)
   unlk   A5
   move.l d7,d1              ; Handle
   LEA    Message(PC),A0     ; Address of Message
   MOVE.L A0,D2              ;  .. to D2
   MOVEQ  #14,D3             ; Length of Message
   JSR    _LVOWrite(A6)      ; Send message
   RTS

Message  dc.b  'Hello, World!',$0a

   end

; Command BLINK TEST.O TO TEST
```

## Listing 4: *LongtoAscii.asm*

```
   IDNT BinAscii
   SECTION ConvAsc,CODE
   XDEF MakeAscii

; Subroutine:  Convert unsigned longword to ASCII string
;   --value received in D0;
;   --result placed in string at (A2)
```

```
; Calling program MUST provide:
;    longword value to be converted in D0;
;    a buffer to hold result string;
;    address of this buffer in register A2;
;    size of this buffer in D2;
; Results:
;    Ascii string, zero-suppressed, in buffer;
;    Register D0 will contain zero if buffer
;    is sufficiently large; non-zero indiates
;    overflow.
; Registers:
;    Standard;  Registers D0, D1, will be changed;
;    all others will be preserved.
;
; First, clear string area to SPACEs.

MakeAscii
     move.l d3,-(a7)
     moveq  #$20,d1          ; space character
     move.w d2,d3
     subq.w #1,d3

Clear
     move.b d1,0(a2,d3.w)
     dbf    d3,Clear
     move.w d2,d3
     subq.w #1,d3

; Calculate digits from low end of number.

DigLoop
     swap   d0               ; high/low word swap
     moveq  #0,d1
     move.w d0,d1            ; high word to D1
     beq.s  Skipper          ; zero, don't divide
     divu   #10,d1           ; divide hi word

Skipper
     move.w d1,d0            ; new hi word value
     swap   d0               ; restore hi/low order
     move.w d0,d1            ; set low value
     divu   #10,d1           ; continue division
     move.w d1,d0            ; new lo word value
     swap   d1               ; get remainder
     or.b   #$30,d1          ; make it Ascii
     move.b d1,0(a2,d3.w)    ; put at low end
     tst.l  d0               ; any more digits?
     dbeq   d3,DigLoop       ; yes, go back
     move.l (a7)+,d3
     rts

     end
```



## Product Review

**Media Line Video and Animation Backgrounds Disk**
Free Spirit Software

suggested retail price $39.95 US

**by Derek Grime**

This two disk set from Free Spirit Software Inc. should find a home in every Amiga desktop video studio. Danish designer and film producer Tore Bahnson has created a series of useful and clever backgrounds that can easily be added to your two and three dimensional animations.

This is a well executed package. All pictures are in the preferred video resolution of 352 x 440. That's overscan, so there are no borders to worry about. They share the same palette as Aegis' VideoScape, allowing them to be easily added to any sequence. They can be compiled into anims or slide shows without having the palette shift from image to image.

Graphics are divided into four catagories. There are directories for text backgrounds, scenic images, structures and high tech pictures. Each category has its own IFF format guide in the parent directory that shows miniaturized versions of each image. This thoughtful addition can be used as a table of contents when you need to find the right background quickly.

The *Scenic* directory contains some wonderful graphics. "Christmas" is especially nice but some, like 'CPU', suffer from interlace shimmer because of the large number of horizontal lines.

*Text* backgrounds are well suited to business presentations. Some are whimsical, like a screen-sized exclamation mark. Others are simple boxes with dropshadows. A couple are graduated crosshatch backgrounds, similar in fashion to those created by TV Text. In interlace mode, they flicker furiously.

*High Tech* gives us a selection of scientific and perspective backgrounds. These are bound to be everyone's favourites. The ever popular perspective grids are here, and they are anti-aliased to give the illusion of being in the highest possible resolution.

*High Tech* also has some well executed illustrations of molecules and human brains. You can almost hear the narration in the background.

*Structures* are a series of intricate pattern-filled backdrops. Some are repeats of the same images with colour changes.

All in all, Media Line's Video and Animation Backgrounds are a dependable collection. These tried and true backgrounds could find use in any studio over and over again.

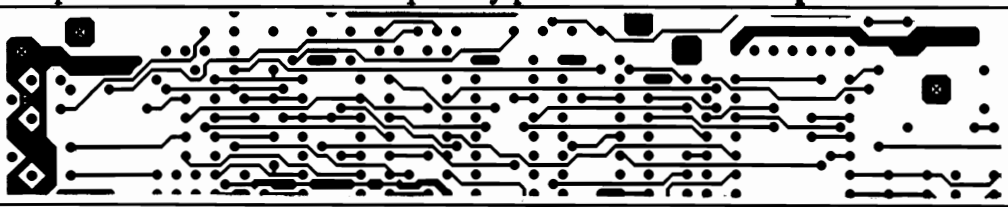
# Black Belt Systems

*is pleased to present*

# Board Master

Here's a product for the electronic engineer, technician, or hobbyist. Priced at only **$99.95**, this Printed Circuit Board layout program that has features that rival those of mainframe PCB CAD systems. Board Master is *extremely* fast, flexible, and reliable. Need quality output of small boards? We can use our HP Laserjet II to generate plots of PCBs up to 4x5 inches at 2:1 for a small fee. Board Master requires 960 k of free memory to run, so you will need an expanded Amiga to use it. Board Master will run on any model Amiga, the memory it requires is mostly fast ram. Lattice 'C' Language source code for our plot drivers is available. Inquire by phone. **Schematic Capture is next!**

- Excellent ARexx interface
- Over 60 PCB ARexx cmds!
- .050 or .025 grid system
- Complete library facility
- Multiple silk screens
- Generates Excellon Drill tapes
- Creates IFF images of PCBs!
- On-screen help, user definable
- Interactive Trace Router -
  - ★ Diagonal or Rectangular
  - ★ Dynamic Trace Sizing
  - ★ Global Trace following
  - ★ Autorouting w/ARexx
- Export hole sizes to silkscreen
- Auto DIP, SIP, and PINGRID
- On-screen plotter simulation
- Global UnDo function
- Extraordinary Macro Capability
- Pan easily with prop gadgets
- Runs on *any* Hard Disk Drive
- NOT COPY PROTECTED!



Actual LaserJet II Output plot segment - support for HPGL, HIPLOT, HP LJII

**Black Belt Systems**
398 Johnson Road
RR1, Box 4272
Glasgow, Montana
59230
406-367-5509

MC-Visa, Check, M.O. C.O.D.
*-or-*
See your Dealer!

The worlds fastest programmers* use the new

# JForth
## Professional 2.0

from

**Delta Research**
Box 1051, San Rafael, CA, 94915, (415) 485-6867
Call or write for information.

JForth is an interactive programming language that compiles directly to 68000 machine code. With JForth, you can compile and interactively test your code in seconds. Our "Clone" feature can generate Royalty Free Applications as small as 3K by automatically removing unused code. Execution speed is comparable to 'C'.

**Language Features**

- 32 bit Forth '83 standard plus '79 & FIG,
- Compile from normal files or BLOCKs,
- Hashing for fast compilation, • Local Variables,
- Source Level Debugger with breakpoints,
- FFP single and IEEE double Floating Point,
- Disassembler plus RPN and Motorola Assemblers
- Object Oriented Dialect, source code included,
- Many utilities: Sort, File I/O, dump structures....

**Amiga Support**

- Call any Amiga library routine by name,
- Complete Amiga structure support plus precompiled include files,
- Easy toolboxes for Graphics & Menus,
- IFF support plus simple animation system with Wipes, Fades, Blits, etc,
- Extensive tutorials and many examples for beginners and experts alike.

* Mike Haas and Phil Burk used JForth to win the Worlds Fastest Programmer Contest, held in Anahiem, California, Nov. 1988.

Ask for JForth at a store near you. Suggested retail price = **$179.95**

# Programming In M2AMIGA

## *Drawing text in colour with style*

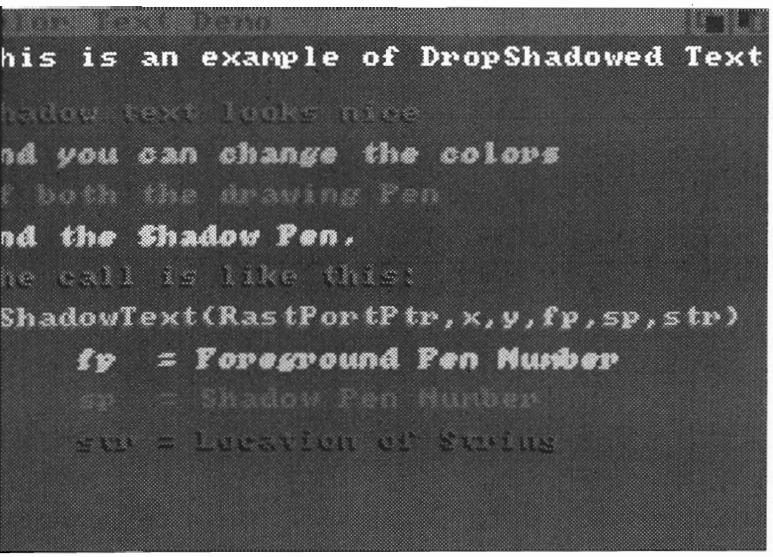
**by Anthony Bryant**

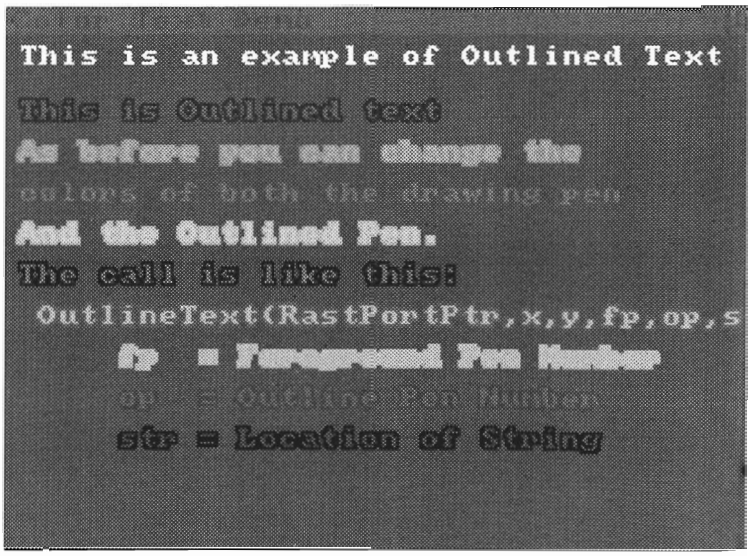
*This is part of a series of articles on using the Modula-2 programming language, specifically the* M2AMIGA *Development System from Interface Technologies.*

Just when you thought there was nothing more to know about using the Amiga Text() graphics function, along comes a programmer with something more to say and a program or two to show. It seems there will always be ever more methods of creating stylish text on the Amiga's hi-res bitmap!

### Technicolour text

Chris Bailey has come up with a neat method of generating technicolour text in a plethora of styles to satisfy even the most jaded designer of games screens. His library of TextFunctions, adapted and presented here as an M2AMIGA library (see *TextFunctions.def* and *TextFunctions.imp* at the end of this article) are simple yet elegant algorithms for overstriking text. As Chris says, "There are probably a lot more things you can do with these routines, so let this be your guide to coming up with more creative text routines, not your sole resource."



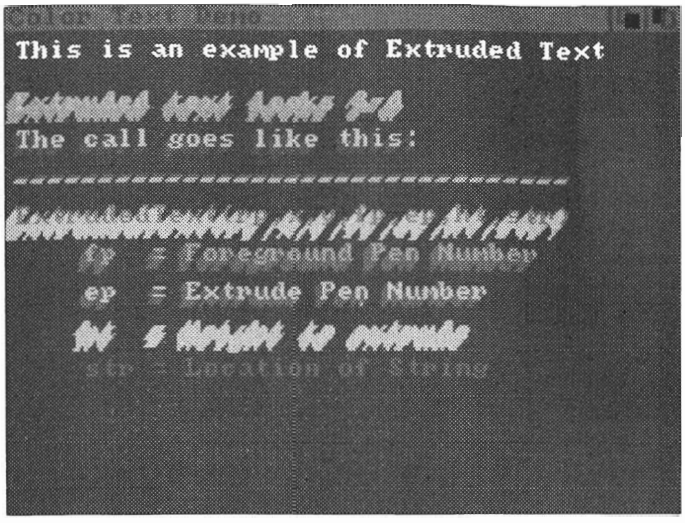

There are five routines in this library, each organized to combine calls to graphics functions: SetDrMd(), SetAPen(), Move() and Text().

The first routine, PROCEDURE DullText(), is simple – plain, ordinary, dull text. All parameters are simply passed on to the appropriate functions, so coordinates *x* and *y* are INTEGER values (as required by Move() ), the *pens* are CARDINAL values (as required by SetAPen() ), and *string* is an ADDRESS (as required by Text() ). Text() requires a null-terminated string (which we can guarantee by passing the string via ADR() ); so we use procedure strlen() to find the length.

The most basic of the text routines is PROCEDURE ShadowText(). It draws a drop-shadow in a darker colour, the *shawdowpen*, and then overstrikes, or *jams*, the same text in a lighter colour, the *frontpen*.

PROCEDURE OutlineText() draws the text with an outline around it, using, what else, an *outlinepen*. Simply offset the position of the text, and overstrike, again. This makes text stand out if you want to put an emphasis on it.

PROCEDURE ExtrudedText() draws a 'trail' leading up to your text, in the *backpen* colour. This can also be described as an extra-long drop-shadow; it gives a 3D look to the text. The length of the 'extruded' part is the *height* parameter, in pixels.

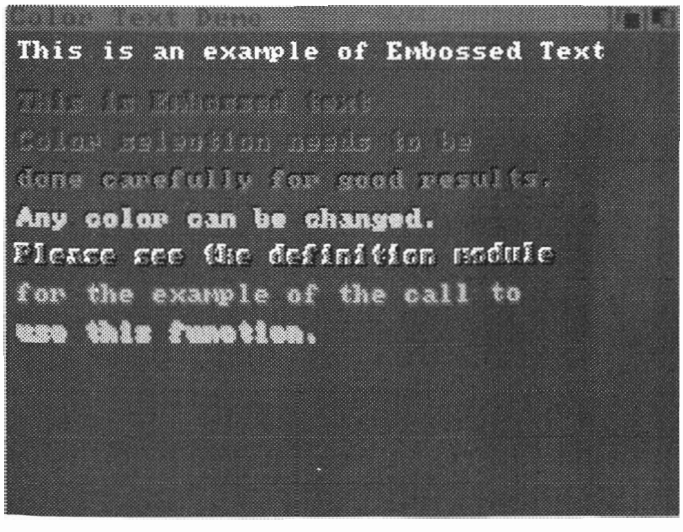



PROCEDURE EmbossedText() is probably the trickiest text function in here. If you want your 'embossed' text to look really good, the most effective method is to use the background colour for the frontpen, using a colour a few notches lighter for *lightpen* and a colour an equal number of notches darker for the *darkpen*. This procedure is dependent on the current palette, so if you want things to look just right you will have to adjust it accordingly. You can get interesting results using other colours, however, including text that looks 'raised' above the common background.

### Technicolourful demo

A sample program showing how to use the procedures contained in the TextFunctions library is provided at the end of this article (see ctdemo.mod).

Since this is a colour text demo, we want to open a low resolution screen and an active window to draw in, with a range of colours to display. Only a few colours are specifically set here, to keep this demo short, but you could also design and add a colour requester to this program, to play with the colour choices. Between displaying each of the four colour text demo screens, the program waits for a press of the left mouse button before advancing.




So let's see more of some spiffy 'technicolourful' text! Chris Bailey's experiments with different algorithms should inspire even more creative effort!

**Listing 1.** TextFunctions.imp

```
IMPLEMENTATION MODULE TextFunctions;

(*
 *  TextFunctions is another easy way to jazz up those boring old
 *    text displays that you might be using in your programs.
 *  There are probably a lot more things you can do with this, so let
 *    this be your guideline to coming up with more creative text!
 *
 *  Original by Chris Bailey, this is freely redistributable
 *
 *  M2Amiga adaptation by Anthony Bryant
 *)


FROM SYSTEM IMPORT
 ADDRESS;

FROM Graphics IMPORT
 RastPortPtr, jam1, Move, Text, SetAPen, SetDrMd;


(* -------------------- strlen subroutine -------------------------- *)

TYPE
  CHARPtr = POINTER TO CHAR;  (* need to operate on string of chars *)

(*  strlen - return length of null-terminated text string
 *)
PROCEDURE strlen(string: ADDRESS): CARDINAL;
 CONST  nul=0C;
 VAR  scan: CHARPtr; count: CARDINAL;
 BEGIN
   scan:= CHARPtr(string); count:=0;
   WHILE (scan^ # nul) DO INC(scan); INC(count); END;
   RETURN (count);
 END strlen;


(*  ---------------- the "TextFunctions" ------------------------------ *)

(*  DullText - just plain old text using specified pen color
 *)
PROCEDURE DullText(rp: RastPortPtr; x,y: INTEGER;
                   frontpen: CARDINAL; string: ADDRESS);
 BEGIN
   SetDrMd(rp,jam1);

   SetAPen(rp,frontpen);
   Move(rp,x,y);
   Text(rp,string,strlen(string));
 END DullText;
```

```
(*  ShadowText - draws a drop-shadow, in a darker color, for your text
 *)
PROCEDURE ShadowText(rp: RastPortPtr; x,y: INTEGER;
                     frontpen,shadowpen: CARDINAL; string: ADDRESS);
 VAR  len: CARDINAL;
 BEGIN
   len:= strlen(string);
   SetDrMd(rp,jam1);

  (*
   * What we do here is simply draw the text in the shadowpen color
   *   offset by a 1 pixel slope. ( 45 degree angle *)
   *
   * And then draw one more in the frontpen color in the original spot.
   *   This makes for a one-pixel shadow behind the text.
   *)

   SetAPen(rp,shadowpen);
   Move(rp,x-1,y+1);
   Text(rp,string,len);

   SetAPen(rp,frontpen);
   Move(rp,x,y);
   Text(rp,string,len);
 END ShadowText;


(*  OutlineText - draws the text with an outline color around it
 *)
PROCEDURE OutlineText(rp: RastPortPtr; x,y: INTEGER;
                      frontpen,outlinepen: CARDINAL; string: ADDRESS);
 VAR  a,b,len: CARDINAL;
 BEGIN
   len:= strlen(string);
   SetDrMd(rp,jam1);

  (*
   * This is kind of neat.  What we do here is draw three sets of three
   *   lines of the same text in the outlinepen color like this:
   *             1 2 3
   *             4 5 6
   *             7 8 9
   * And then draw one more in the frontpen color in the '5' spot.
   *   This makes for a one-pixel outline around the text.
   *)

   SetAPen(rp,outlinepen);
   FOR a:= x-1 TO x+1 DO
     FOR b:= y-1 TO y+1 DO
       Move(rp,a,b);
       Text(rp,string,len);
     END;
   END;

   SetAPen(rp,frontpen);
   Move(rp,x,y);
   Text(rp,string,len);
 END OutlineText;
```

```
(*  ExtrudedText - draws an extra-long drop-shadow for a 3-D look.
 *)
PROCEDURE ExtrudedText(rp: RastPortPtr; x,y: INTEGER;
                       frontpen,backpen,height: CARDINAL; string: ADDRESS);
 VAR  a: INTEGER; len: CARDINAL;
 BEGIN
   len:= strlen(string);
   SetDrMd(rp,jam1);

  (*
   * What we do here is draw the same text at numerous offsets
   *   according to the 'height' value, moving in a 45 degree
   *   slope, in order to "slew" the text.
   *
   * And then draw one more in the frontpen color in the original spot.
   *   This variation on the drop-shadow style creates a 3-D look.
   *)

   SetAPen(rp,backpen);
   FOR a:= 1 TO INTEGER(height)  DO
     Move(rp,x-a,y+a);
     Text(rp,string,len);
   END;

   SetAPen(rp,frontpen);
   Move(rp,x,y);
   Text(rp,string,len);
 END ExtrudedText;


(*  EmbossedText - makes the text look .. well..er.. 'embossed'
 *)
PROCEDURE EmbossedText(rp: RastPortPtr; x,y: INTEGER;
                       frontpen,lightpen,darkpen: CARDINAL; string: ADDRESS);
 VAR  a,b: INTEGER; len: CARDINAL;
 BEGIN
   len:= strlen(string);
   SetDrMd(rp,jam1);

  (*
   * This is kind of similar to what we were doing with the OutlineText
   *   routine.  Three areas need to be drawn into, with F=frontpen,
   *   L=lightpen, and D=darkpen, so the algorithm does something like this:
   *
   *    STEP 1          STEP 2          STEP 3
   *      DD              DD              DD
   *      DD              DLL             DFL
   *                       LL              LL
   *
   * As you can see, we come out with a embossed effect.  This
   *   is not the exact order things are drawn in, but it gives you
   *   a general idea of what needs to be done to achieve the
   *   embossed effect.
   *)

   SetAPen(rp,darkpen);
   FOR a:= 0 TO 1 DO
     FOR b:= 0 TO 1 DO
       SetAPen(rp,darkpen);
       Move(rp,x-a,y-b);
       Text(rp,string,len);
```

```
      SetAPen(rp,lightpen);
      Move(rp,x+a,y+b);
      Text(rp,string,len);
    END;
  END;

  SetAPen(rp,frontpen);
  Move(rp,x,y);
  Text(rp,string,len);
END EmbossedText;


END TextFunctions.imp
```

## Listing 2.: *TextFunctions.def*

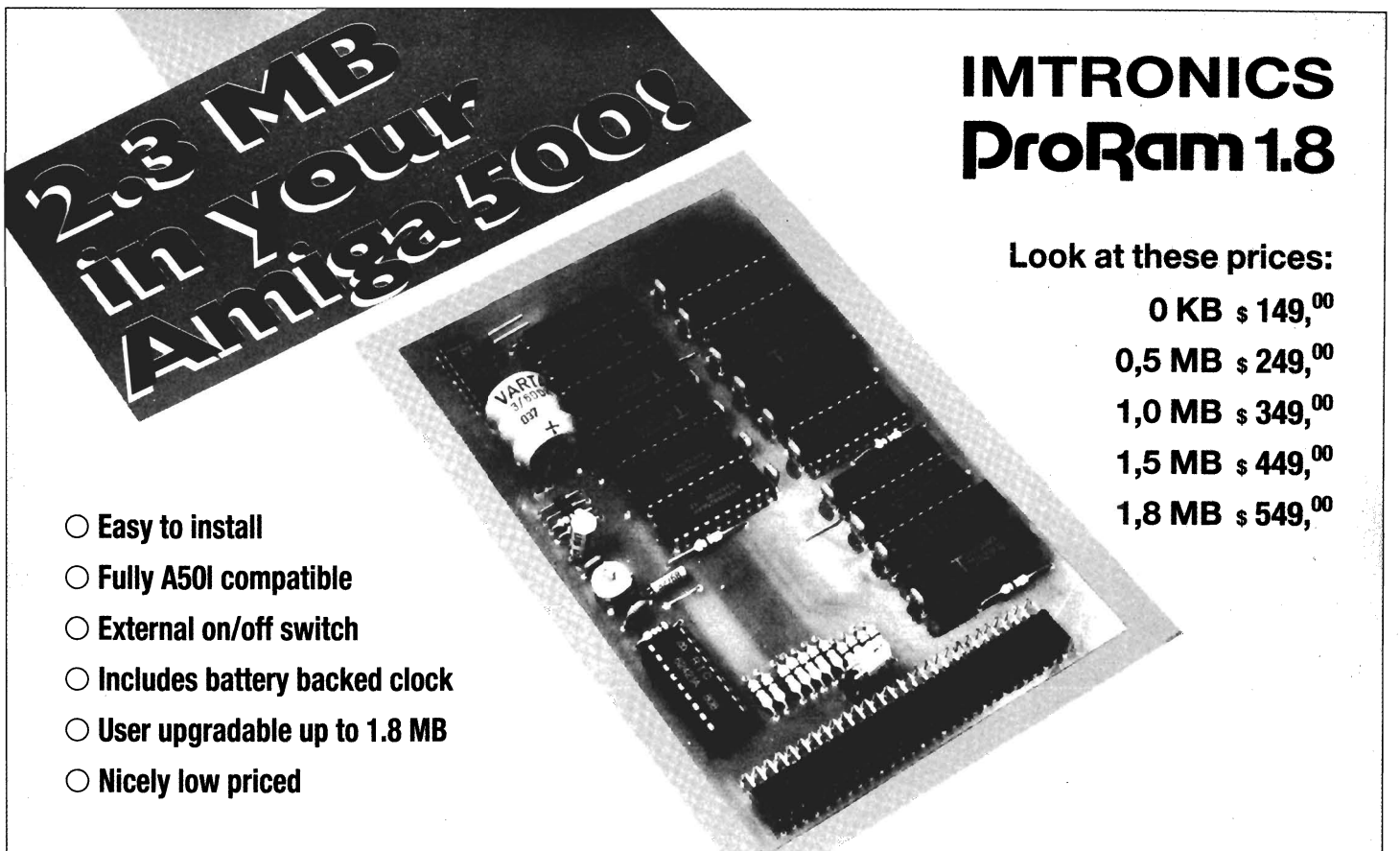
```
DEFINITION MODULE TextFunctions;

(*
 *   TextFunctions is another easy way to jazz up those boring old
 *     text displays that you might be using in your programs.
 *   There are probably a lot more things you can do with this, so let
 *     this be your guideline to coming up with more creative text.
 *
 *   Original by Chris Bailey, this is freely redistributable
 *
 *   M2Amiga adaptation by Anthony Bryant
 *)

FROM SYSTEM IMPORT
 ADDRESS;

FROM Graphics IMPORT
 RastPortPtr;


(*
 * NOTE:  All text routines use Amiga Graphics functions:
 *        Move() - to moveto x,y pixels, and
 *        Text() - to draw the text in a null-terminated
 *                 string of characters.
 *)


(*  DullText - just plain old text, using specified pen color
 *)
PROCEDURE DullText(rp: RastPortPtr; x,y: INTEGER;
                   frontpen: CARDINAL; string: ADDRESS);

(*  ShadowText - draws a drop-shadow for your text
 *)
PROCEDURE ShadowText(rp: RastPortPtr; x,y: INTEGER;
                     frontpen,shadowpen: CARDINAL; string: ADDRESS);

(*  OutlineText - draws the text with an outline around it
 *)
PROCEDURE OutlineText(rp: RastPortPtr; x,y: INTEGER;
                      frontpen,outlinepen: CARDINAL; string: ADDRESS);

(*  ExtrudedText - draws an extra-long drop-shadow for a 3-D look.
 *)
PROCEDURE ExtrudedText(rp: RastPortPtr; x,y: INTEGER;
                       frontpen,backpen,height: CARDINAL; string: ADDRESS);

(*  EmbossedText - makes the text look .. well..er.. 'embossed'
 *)
PROCEDURE EmbossedText(rp: RastPortPtr; x,y: INTEGER;
                       frontpen,lightpen,darkpen: CARDINAL; string: ADDRESS);


END TextFunctions.def
```

## Listing 3.: *ctdemo.mod*

```
MODULE ctdemo;   (* Testing the TextFunction routines *)

(*  M2Amiga demo program written by Anthony Bryant  *)

FROM SYSTEM IMPORT
 ADR, ADDRESS;

FROM Arts IMPORT
 TermProcedure;

FROM Exec IMPORT
 WaitPort, GetMsg, ReplyMsg;

FROM Intuition IMPORT
 WindowFlags, WindowFlagSet, IntuiMessagePtr, IDCMPFlags, IDCMPFlagSet,
 WindowPtr, ScreenPtr, customScreen, NewScreen, NewWindow, selectDown,
 OpenWindow, OpenScreen, CloseWindow, CloseScreen;

FROM Graphics IMPORT
 RastPortPtr, SetRast, SetRGB4, ViewModeSet, ViewPortPtr;

FROM TextFunctions IMPORT
 DullText, ShadowText, OutlineText, ExtrudedText, EmbossedText;


VAR     (* our global pointers *)
  wp: WindowPtr;
  sp: ScreenPtr;
  rp: RastPortPtr;
  vp: ViewPortPtr;


(* InitScreen - open a custom lo-res screen for this demo
 *)
PROCEDURE InitScreen();
 VAR
   ns: NewScreen;  (* temporary structure *)
 BEGIN
   WITH ns DO
     leftEdge:= 0; topEdge:= 0;
     width:= 320; height:= 200; depth:= 5;
     detailPen:= 0; blockPen:= 1;
     viewModes:= ViewModeSet{};  (* lo-res *)
     type:= customScreen;
     font:= NIL;
     defaultTitle:= ADR("Color Text Demo");
```

```
    gadgets:= NIL;
    customBitMap:= NIL;
  END;
  sp:= OpenScreen(ns);
END InitScreen;


(* InitWindow - open an active backdrop window to draw in
 *)
PROCEDURE InitWindow();
 VAR
   nw: NewWindow;  (* temporary structure *)
 BEGIN
   WITH nw DO
     leftEdge:= 0; topEdge:= 0;
     width:= sp^.width; height:= sp^.height;  (* same as screen *)
     detailPen:= 0; blockPen:= 1;
     title:= NIL;
     flags:= WindowFlagSet{activate,rmbTrap,backDrop};
     idcmpFlags:= IDCMPFlagSet{mouseButtons};
     type:= customScreen;
     firstGadget:= NIL;
     screen:= sp;
     bitMap:= NIL;
   END;
   wp:= OpenWindow(nw);
 END InitWindow;


(* SetScreenColors - for the technicolor text !
 *   NOTE: only a few screen colors are set, in order to have at least
 *   two good looking examples in the 'Embossed' section of the demo !
 *)

PROCEDURE SetScreenColors();
 BEGIN
   SetRGB4(vp, 0, 5, 5, 5);  (* Dark Grey - Background *)
   SetRGB4(vp, 1, 7, 7, 7);  (* Med. Grey              *)
   SetRGB4(vp, 2, 3, 3, 3);  (* Near Black             *)
   SetRGB4(vp,30, 7, 9, 9);  (* Med. Green             *)
   SetRGB4(vp,31, 3, 5, 5);  (* Dark Green             *)
 END SetScreenColors;


(* Wait for mouse SELECT button click DOWN to advance
 *)
PROCEDURE WaitForClick();
 VAR
   msg: IntuiMessagePtr; class: IDCMPFlagSet; code: CARDINAL;
 BEGIN
   LOOP  (* now sleep and wait for a click *)

     WaitPort(wp^.userPort);
     msg:= IntuiMessagePtr(GetMsg(wp^.userPort));
     IF (msg # NIL) THEN
       class:= msg^.class;
       code:= msg^.code;
       ReplyMsg(ADDRESS(msg));
       IF (mouseButtons IN class) AND (code = selectDown) THEN EXIT; END;
     END;
   END;  (* LOOP *)
 END WaitForClick;


(* ---------------- Now for the "Demos" ! ------------------------- *)

(* DoShadowText - draw some color text in the drop-shadow style
 *)
PROCEDURE DoShadowText();
 BEGIN
   SetRast(rp, 0);  (* clear screen *)
   DullText(rp,5,20,7,ADR("This is an example of DropShadowed Text"));

   ShadowText(rp,5, 40, 1, 2,ADR("Shadow text looks nice"));
   ShadowText(rp,5, 55, 6, 3,ADR("And you can change the colors"));
   ShadowText(rp,5, 70,24,12,ADR("Of both the drawing Pen"));
   ShadowText(rp,5, 85,28,19,ADR("And the Shadow Pen."));
   ShadowText(rp,5,100,30, 4,ADR("The call is like this:"));
   ShadowText(rp,5,115,27, 9,ADR(" ShadowText(RastPortPtr,x,y,fp,sp,str)"));
   ShadowText(rp,5,130,11,10,ADR("     fp  = Foreground Pen Number"));
   ShadowText(rp,5,145, 5,17,ADR("     sp  = Shadow Pen Number"));
   ShadowText(rp,5,160,12,18,ADR("     str = Location of String"));

   WaitForClick();
 END DoShadowText;


(* DoOutlineText - draw some color text in the outline style
 *)
PROCEDURE DoOutlineText();
 BEGIN
   SetRast(rp, 0);  (* clear screen *)
   DullText(rp,5,20,7,ADR("This is an example of Outlined Text"));

   OutlineText(rp,5, 40, 1, 2,ADR("This is Outlined text"));
   OutlineText(rp,5, 55, 6, 3,ADR("As before you can change the"));
   OutlineText(rp,5, 70,24,12,ADR("colors of both the drawing pen"));
   OutlineText(rp,5, 85,28,19,ADR("And the Outlined Pen."));
   OutlineText(rp,5,100,30, 4,ADR("The call is like this:"));
   OutlineText(rp,5,115,27, 9,ADR(" OutlineText(RastPortPtr,x,y,fp,op,str)"));
   OutlineText(rp,5,130,11,10,ADR("     fp  = Foreground Pen Number"));
   OutlineText(rp,5,145, 5,17,ADR("     op  = Outline Pen Number"));
   OutlineText(rp,5,160,12,18,ADR("     str = Location of String"));

   WaitForClick();
 END DoOutlineText;


(* DoExtrudedText - draw some color text in the 3-D style
 *)
PROCEDURE DoExtrudedText();
 BEGIN
   SetRast(rp, 0);  (* clear screen *)
   DullText(rp,5,20,7,ADR("This is an example of Extruded Text"));

   ExtrudedText(rp,5, 40, 6, 3, 5,ADR("Extruded text looks 3-d"));
   ExtrudedText(rp,5, 55,10,12, 3,ADR("The call goes like this:"));
   ExtrudedText(rp,5, 70, 9,38, 2,ADR("--------------------------------"));
   ExtrudedText(rp,5, 85,24,11, 6,ADR("ExtrudedText(rp,x,y,fp,ep,ht,str)"));
   ExtrudedText(rp,5,100, 3,23, 4,ADR("    fp  = Foreground Pen Number"));
   ExtrudedText(rp,5,115,27, 9, 3,ADR("    ep  = Extrude Pen Number"));
```

```
  ExtrudedText(rp,5,130,11,10, 5,ADR("    ht  = Height to extrude"));
  ExtrudedText(rp,5,145, 5,17, 4,ADR("    str = Location of String"));

  WaitForClick();
 END DoExtrudedText;


(* DoEmbossedText - draw some text in an embossed style
 *)
PROCEDURE DoEmbossedText();
 BEGIN
  SetRast(rp, 0);  (* clear screen *)
  DullText(rp,5,20,7,ADR("This is an example of Embossed Text"));

  EmbossedText(rp,5, 40, 0, 1, 2,ADR("This is Embossed text"));
  EmbossedText(rp,5, 55, 0,30,31,ADR("Color selection needs to be"));
  EmbossedText(rp,5, 70,24,12, 4,ADR("done carefully for good results."));
  EmbossedText(rp,5, 85,28,19, 8,ADR("Any color can be changed."));
  EmbossedText(rp,5,100,30, 4, 7,ADR("Please see the definition module"));
  EmbossedText(rp,5,115,27, 9, 5,ADR("for the example of the call to"));
  EmbossedText(rp,5,130,11,10, 6,ADR("use this function."));

  WaitForClick();
 END DoEmbossedText;


(* ----------------- now, let's get on with the show ----------------- *)

PROCEDURE Cleanup();   (* this will be our exit routine *)
 BEGIN
  IF (wp # NIL) THEN CloseWindow(wp); END;
  IF (sp # NIL) THEN CloseScreen(sp); END;
 END Cleanup;


BEGIN
 TermProcedure(Cleanup);  (* as always, leave a tidy Amiga *)

 InitScreen();
 IF (sp # NIL) THEN  (* proceed *)

  vp:= ADR(sp^.viewPort);  (* SetRGB4() needs a viewport *)

  SetScreenColors();  (* must set up some colors, otherwise random ! *)

  InitWindow();
  IF (wp # NIL) THEN  (* proceed further ! *)

   rp:= wp^.rPort;  (* Move() and Text() functions need a rastport *)

   (* the four demos that you get to see - just click to advance *)
   DoShadowText();
   DoOutlineText();
   DoExtrudedText();
   DoEmbossedText();

  END; (* valid window *)
 END;  (* valid screen *)

 (* TermProcedure kicks in here closing up everything ! *)

END ctdemo.mod
```

# Toy: A Small Compiler For The Amiga

## *Part 4: Code generation and optimization*

**by Chris Gray**

*Chris Gray is a programmer from Edmonton, Alberta. He writes: "I am currently working for Myrias Research Corporation, which is building an easy-to-use parallel supercomputer. Our first regular customer machine was shipped May 1989, and we look forward to giving the big boys like Cray and NEC a run for their money. I fell in love with the Amiga when it was first announced, and bought one of the first in the city. Since then I've been Amiga programming steadily. My major software releases, both shareware, are the Draco compiler, along with its accompanying libraries and tools, and Amiga Empire."*

In many ways, the most interesting part of a compiler is the code generator. The previous parts of the Toy compiler that we have seen have all been mostly independent of the nitty-gritty stuff of actually generating the desired machine code. If we are generous in our definition, code generation can also include all of the optimizations, which, to many, are the only really interesting parts of a compiler. Unfortunately, Toy is a very simple compiler, and doesn't really have any optimization. The couple of examples shown are just that: examples.

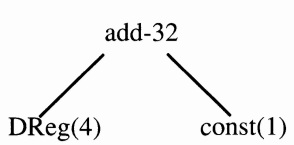
I will begin with a simple discussion of some of the alternatives in code generation. Next I will briefly discuss the code in Toy's code generator (which is very straightforward). This is followed by a run-through of generating code for a simple example, and by a short discussion of some optimizations.

### Alternatives

As with most parts of the compiler, there are several ways to approach the task of code generation. In a desire to avoid a lot of hand-written code, and to keep the code generator as independent of the target CPU as possible, many modern compilers use a table-driven approach. This requires that the parsed program be represented in some sort of tree form.

The table entries contain a coded representation of a subtree, along with the assembler instructions to be generated, with appropriate substitutions from the subtree. As an example, suppose a compiler had an expression node that looked like this:

```
        add-32
       /      \
DReg(4)        const(1)
```



This represents the desire to do a 32-bit add of a value in a data register and a constant. The value is in register 4, and the constant to be added is '1'. An MC68000 code generator might have table entries somewhat like this:

```
node    op1    op2                  asm string      size
-------------------------------------------------------
add-32  A  const(1..8)             "addq.l #%1,A%2"   2
add-32  D  const(1..8)             "addq.l #%1,D%2"   2
add-32  A  const()                 "adda.l #%1,A%2"   6
add-32  D  const()                 "add.l #%1,D%2"    6
add-32  A  const(-32768..32767)    "lea %1(A%2),A%2"  4
```

where the fields are:

**node**: node type to match
**op1**: operand #1 (here a register of type A or D)
**op2**: operand #2 (here a numeric constant, perhaps restricted)
**asm string**: assembler string to generate
**size**: size in bytes

The operand fields are numbered, allowing their values to be substituted into the assembler string. Our sample node above would thus match both the second and fourth entries. The compiler could correctly generate either one, but since the second is four bytes shorter, it would be preferred. In reality, a compiler for the 68000 family would likely encode the usability of many addressing modes in the same table entry, and also encode the operand sizes. Large tables are usually left, however, and the compiler spends a lot of time deciding which entries to use.

Note that the above table entries contained a string for the emitted assembler code. This implies that such a compiler would produce assembler source that would have to be assembled after the compilation phase. Aztec C for the Amiga operates in this fashion, as do many C compilers based on the original *pcc* (portable C compiler). Different table entries could be used to build machine code directly. A distinct advantage of table-driven code generation is that, given a sufficiently general representation, only the table need be changed to generate code for a new CPU; the table-matcher and associated code stays the same.

One of the critical resources in today's CPUs is the register set. Good code requires good use of the available registers. As many

variables as possible should be in registers (unless, of course, doing so results in poorer code!). The reason for this is quite simple: registers can be addressed directly, without an explicit address, thus saving instruction bytes. They are also located in the CPU chip itself, so no memory cycles are needed to load or store them. Compilers need some scheme whereby registers are matched up with values and variables that are being used. One of the best schemes is 'register colouring', which 'paints' each value with the colour (number) of the register that it will be in. A rule here, of course, is that no two values can have the same colour at the same time. This type of scheme ignores any 'register' specification that the programmer gave, since it can usually do a better job anyway. Another common scheme is to have a register allocator which allocates registers to values as needed, and then does 'spilling' to the stack when the registers run out and more are needed. The latter is what is done in Draco, and partially in Toy (although without spilling). There are of course many important details that I am ignoring (such as register colouring taking a fairly long time as it traverses the graph structures built to represent the program).

Closely related to register allocation is storage allocation. Where do all of the variables go? For a 68000 family CPU, the local variables and parameters are normally in registers or on the stack. Global variables are in locations fixed when the program is loaded, and are referenced using full 32-bit absolute addresses (the so-called 'large' model for the 68000) or using 16-bit offsets from a base register (the 'small' model). Both of the commercial C compilers allow you to choose which of these models you want to use. The first allows as much global variable space as you want, but is bulkier and slower. The second is faster but limits you to 64K worth of global variables, and ties up a critical address register. Both Toy and Draco allow only the large model. Modifying Toy to use the small model for global variables should not be difficult – interested programmers are encouraged to try it for themselves.

## Code generation in Toy

Toy's code generator is shown in Listing 1, file 'codeGen.d'. It is a fairly long file, but has no complex routines in it. In the following description, I will assume some familiarity with MC68000 assembler language, but not with the detailed format of machine code. When generating code, it is useful to classify the target instructions so that a common interface can be used. The 68000 doesn't fit perfectly into such a classification, but it is still useful. For the purposes of Toy, I have split the instructions up as follows:

**opSpecial**: those that don't fit any other form.
**opSingle**: operations on a single operand. The operand can be any size (byte, word, long) and is addressed by any addressing mode that yields modifiable data.
**opEA**: these are very similar to 'opSingle', except that no explicit size is given, only an effective address.
**opRegister**: these contain an effective address and a register number; they have two operands.
**opModed**: these contain a register, an effective address, and an op-mode specifying size and direction of operation.
**opMove**: these are the MOVE instructions. They have two effective addresses.
**opBranch**: these are PC-relative branches, which may be conditional or unconditional.

In general (with many exceptions) an MC68000 instruction can be split up into bit-fields:

4-bit primary op-code
3-bit register number
3-bit op-mode
6-bit effective address

Many individual instructions treat some of the fields as additional op-code bits, and many use only the bottom two bits of the op-mode as a size indicator.

The many constants defined at the beginning of "codeGen.d" are straight out of the MC68000 reference manuals. They include such things as the range of data registers that Toy will use, names for the stack pointer and frame pointer (more on that later), names for the various addressing modes that the CPU has, names for the operand sizes that are available, and names for the opcodes that Toy will use.

Procedure *genWord* is used for all emitted code. It adds a 16-bit word to the generated instruction stream. If the fixed-size buffer (here 10000 bytes) overflows, it aborts the compilation. The various *opXXX* routines use *genWord* to emit specific classes of instructions. They build the instruction word by shifting the various fields into their proper places. This level of indirection may seem pointless at first sight, but this classification of instructions and the use of a common generation point is very useful when adding a peephole optimizer.

Next come all of the routines we have seen called from the parser. I won't describe them in detail, since they are all straightforward once the overall context of code generation is known. Using registers is simple, but where *exactly* are the various variables and parameters going to be? Global variables will be stored in a contiguous chunk of memory allocated by AmigaDOS when the Toy program is loaded. In AmigaDOS terms, they will be in a BSS hunk. For simplicity, each variable occupies two bytes, so we need only add two to a global counter for each global variable declared. This gives us the offset of the next global variable in the one chunk (the first being at offset 0). The offset is stored in the variable's *se_value* field. When a function with parameters is called, we will set up the stack as shown in Figure 1.

What does all of this mean, and how does it happen? 'FP' stands for 'frame pointer', and is a register that we will use to contain the address of the above 'stack frame' for this procedure call. Note that it points directly at the local variables, so they can be accessed via an offset from the frame pointer. Also, there are a fixed number of bytes (eight) between where the frame pointer points and the parameters, so they too can be accessed via the frame pointer (A6).

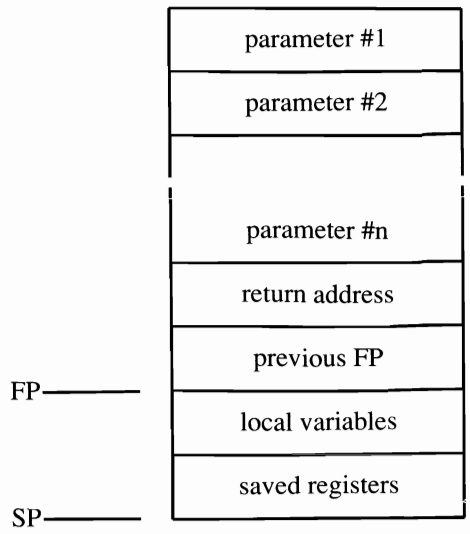



*Figure 1: Stack diagram for function call with parameters*

Recall that the procedure parser called *allocateVariable* for each parameter as they were encountered. It would then be easy to assign offsets 0, 2, 4, 6, etc. to the parameters. From the above diagram, however, that would be incorrect. The last parameter is at offset 8, the next at offset 8 + 2, the next 8 + 2 * 2, etc. and the first is at offset 8 + N * 2, where N is the number of parameters, which is not known until all of the parameters are counted. This is the reason for the *fixLocals* call that is made by *procEndLocals*: it adds "ParameterSize + 8" to the *se_value* field of all parameters and local variables, which gives them their correct frame pointer offset.

Assuming an empty stack, and recalling that the MC68000 stack grows downwards (pushing something decreases the address in the stack pointer), the sequence of events in creating a stack frame is:

- push first parameter
- push second parameter
- ...
- push nth parameter
- do a JSR to the procedure, which pushes the return address
- do a LINK #M,FP instruction, which pushes the current value of FP, copies SP into FP, then adds *M* (a negative number, to allocate the space for the local variables) to SP
- save registers by pushing them onto the stack

The operation of *allocateVariable*, *procStartParameters*, *procEndParameters*, *procEndLocals* and *codeInit* should now be fairly clear. *codeTerm* generates code to reverse the actions done by the entry code:

- restore registers by popping them from the stack
- do an UNLK FP to remove the space for the local variables and pop the previous value of FP
- pop the return address into a temporary (A0)
- increment SP to remove the parameters
- return, by jumping indirect through A0

Here we see our first 'optimization'. If there are no parameters to this procedure, the last three steps above can be shortened into just emitting a simple RTS instruction. The MC68010/20/30/40 all have an RTD ('return and deallocate') instruction, which is able to do the return and parameter popping in one instruction. This could be generated in response to an appropriate compiler flag. The calls to *codeStartProc* and *codeEndProc* are to the system interface. They ready it for a new function whose name is given, and then request that it save away (or write directly) the code that was generated for it.

Variable *NextRegister* is used to implement a small stack of registers for expression evaluation. It starts out at D_REG_FIRST (D7), which is the highest numbered data register. It can go as low as D_REG_LAST (D2) for usable registers. Allocating a register consists of decrementing *NextRegister*. If it is below D_REG_LAST and a register is needed, our small stack has overflowed, and the compiler aborts. The standard fix for this situation is to 'spill' registers onto the stack. For example, if *NextRegister* equals D_REG_LAST - 1 (D1) when a register is needed, move the contents of D7 to the stack and reuse D7. It is then necessary to make sure that all needed register values are actually in the registers when an instruction using them is generated. Draco uses a routine called *needRegs* for this purpose.

## Example of code generation

As an example, we will return to the small program fragment given in the first article in this series:

```
int N, Sum;

proc main()void:

   Sum := 0;
   readln(N);
   while N ~= 0 do
      Sum := Sum + N;
      readln(N);
   od;
   write("The sum is: ", Sum, "\n");
corp;
```

The calls to the code generator for this file go like this:

```
Call                      effect/notes
-------------------------------------------------------
setGlobalSize(276)        space used by run-time-system
                          (ignored in the rest of this)

allocateVariable(<N>)     se_value := 0;
                          globalSize := 2

allocateVariable(<Sum>)   se_value := 2;
                          globalSize := 4

procStartParameters()     parSize := 0
```

```
procEndParameters()        localSize := 8

procEndLocals()            fixLocals(8); localSize := 0

codeInit(<main>)           LINK     #0,FP
                           MOVEM.L <D2-D7>,-(SP)
                           NextRegister := 7

pushConstant(0)            CLR.W    D7
                           NextRegister := 6

popVariable(<Sum>)         NextRegister := 7
                           MOVE.W   D7,global_at_0

callSpecial(rc_readLine)   JSR      _toy_readLine

callSpecial(rc_readInt)    JSR      _toy_readInt
                           MOVE.W   D0,D7
                           NextRegister := 6

popVariable(<N>)           NextRegister := 7
                           MOVE.W   D7,_global_at_2

LOOP_POSITION := getPos()  LOOP_POSITION := 0x24

pushVariable(<N>)          MOVE.W   _global_at_0,D7
                           NextRegister := 6

pushConstant(0)            CLR.W    D6
                           NextRegister := 5

compareValues(<~=>)        NextRegister := 6
                           CMP.W    D6,D7
                           SNE      D7
                           ANDI.W   #1,D7

BRANCH_POSITION := branchForward(true)
                           NextRegister := 7
                           TST.W    D7
                           BEQ.W    ???
                           BRANCH_POSITION := 0x38

pushVariable(<Sum>)        MOVE.W   _global_at_2,D7
                           NextRegister := 6

pushVariable(<N>)          MOVE.W   _global_at_0,D6
                           NextRegister := 5

binaryOp(<+>)              NextRegister := 6
                           ADD.W    D6,D7

popVariable(<Sum>)         NextRegister := 7
                           MOVE.W   D7,_global_at_2

callSpecial(rc_readLine)   JSR      _toy_readLine

callSpecial(rc_readInt)    JSR      _toy_readInt
                           MOVE.W   D0,D7
                           NextRegister := 6

popVariable(<N>)           NextRegister := 7
                           MOVE.W   D7,_global_at_0

branchTo(LOOP_POSITION)    BRA.S    offset 0x24

patchBranch(BRANCH_POSITION)
                           branch at 0x36 to 0x64

emitString("The sum is: ")
                           JSR      offset 0x76
                           "The sum is :"

callSpecial(rc_printString)
                           JSR      _toy_printString

pushVariable(<Sum>)        MOVE.W   _global_at_2,D7
                           NextRegister := 6

callSpecial(rc_printInt)   NextRegister := 7
                           MOVE.W   D7,-(SP)
                           JSR      _toy_printInt

emitString("\n")           JSR      offset 0x90
                           "\n"

callSpecial(rc_printString)
                           JSR      _toy_printString

codeTerm(<main>)           MOVEM.L (SP)+,<D2-D7>
                           UNLK     FP
                           RTS
```

Four very minor optimizations have occurred here. Two entail using a CLR instruction to put a zero into a register. The third is using the RTS for procedure exit instead of the longer sequence. The last is using a short backwards branch in the 'while' loop. Note the use of a JSR around a string constant: it places the address of the string constant onto the stack, which is precisely where it is needed when calling *_toy_printString*. This trick saves having a separate buffer somewhere for string constants. Unfortunately, if confuses the heck out of disassemblers and debuggers!

## Optimizations

We have seen examples of the 'optimizations' in the Toy compiler. They are just special cases where it is obvious that a better code sequence exists. Others could be added (such as using a MOVEQ in *pushConstant* for values from -128 to +127). Doing this sort of thing makes the code better, but it doesn't fix any of the big problems. The most glaring problem in Toy is the code for a comparison. This was the simplest way to allow both boolean variables and comparisons to work the same in conditional branches. Generating a value in a register is not necessary unless the value is being assigned to something. Where Toy currently generates (from the above sequence):

```
CMP.W   D6,D7
SNE     D7
ANDI.W  #1,D7
TST.W   D7
BEQ.W   label
```

we would like to see something like:

```
CMP.W   D6,D7
BEQ     label
```

Notice that the condition (NE on the Scc, EQ on the Bcc) has been reversed. This is because, on a 'while' statement (or on

an 'if' statement for that matter), we branch around the body or alternative if the condition is *not* met. Since there can be only one conditional branch being generated at a time, we could store the condition when *compareValues* is called, and generate the proper conditional branch when *branchForward* is called. We would also need special code in *popVariable* to generate the *bool* value before trying to store it to a variable. (If this isn't clear, consider what would happen if this optimization were done and we had a statement like "flag := a < b;".)

The next most obvious problem is that all operations are done only in registers. The 68000 is quite capable of adding a value in a variable directly into a register, for instance. In Draco, this sort of thing is handled in two ways. The first has the compiler carrying around explicit descriptions of values, so it can generate a memory operation when appropriate. The second involves fixes in the peepholer. MOVE instructions on the 68000 can have two effective addresses, so they can move a value from one variable (or a constant) directly to another, without using a register at all. Also available are immediate-mode instructions that can operate on a memory operand and a constant. Applying these to the code for the above sample program, we could get:

```
    link.w      fp,#0
    movem.l     <d2,d3,d4,d5,d6,d7>,-(sp)
    clr.w       Sum
    jsr         _toy_readLine
    jsr         _toy_readInt
    move.w      d0,N
LOOP:
    tst.w       N
    beq         LOOPEXIT
    move.w      Sum,d7
    add.w       N,d7
    move.w      d7,Sum
    jsr         _toy_readLine
    jsr         _toy_readInt
    move.w      d0,N
    bra         LOOP
LOOPEXIT:
    jsr         16(pc)
    "The sum is: "
    jsr         _toy_printString
    move.w      Sum,-(sp)
    jsr         _toy_printInt
    jsr         4(pc)
    "\n"
    jsr         _toy_printString
    movem.l     (sp)+,<d7,d6,d5,d4,d3,d2>
    unlk        fp
    rts
```

This, with minor differences (the LINK/UNLK are not there since there are no local variables, the TST of *N* is done with a MOVE to D7, the strings are at the end of the code and hence referenced with a PEA, and only registers D7 and D6 are saved/restored) is the code that Draco generates. So, we must regard all of these as 'easy' optimizations.

A 'medium' optimization is a good peephole optimizer. This technique is so named because it views the generated code through a small 'peephole', and simply replaces a short sequence of instructions with another that is equivalent, but shorter/faster. A good peepholer can make up for a poor code generator. I prefer not to overload the peepholer, however, since it is called upon for every instruction generated. For example, many people consider changing multiply/divide by a power of two into a shift (signed divide is a bit different) to be a peepholer's job. I consider it to be a special case that should be done earlier. The peepholer is typically the place where the CPU's 'funny' instructions or modes can be used, or where unfortunate code sequences, which occur because of the design of the code generator, can be fixed up.

An example of the latter for Draco are the *opModed* instructions, which take two operands. For complex expressions, the operations are always done from memory to register, but in some simple cases, the register to memory variants will work. For example, in:

```
register int i;
int j;

j := j + i;
```

the default code generated is something like:

```
move.w      j,d7
add.w       reg_of_i,d7
move.w      d7,j
```

but the peepholer can change it to:

```
add.w       reg_of_i,j
```

Other peepholes, involving the 68000's auto-increment addressing mode, can yield considerable improvements. A frustrating aspect of many optimizations is that they affect each other. For example, in the fragment above, the difference is that in the optimized case the value of *j* is not in a register at the end. If the very next code used *j*, the first sequence could be better.

Other 'medium' optimizations are register-tracking and condition-tracking. Here the compiler keeps track of what will be in the scratch registers and in the CPU's condition codes throughout the code generation. If a value is required that is already available, it is just used directly, rather than being refetched or recomputed.

'Hard' optimizations include things like register colouring, automatically assigning variables to registers, code motion (moving a piece of code out of a loop since it always gets the same result, for example), strength reduction (such as changing a bunch of array indexing operations into pointer increments and dereferences), dead code elimination (removing code that cannot be executed), and so on.

Applying these sorts of things, a good compiler could turn our example into:

```
    move.w    d2,-(sp)          ; only low 16 bits used
    clr.w     d2
    bra       ENTRY
LOOP:
    add.w     d0,d2
ENTRY:
    jsr       _toy_readLine
    jsr       _toy_readInt
    tst.w     d0
    bne       LOOP              ; runs faster this way
    pea       STRING1
    jsr       _toy_printString
    move.w    d2,-(sp)
    jsr       _toy_printInt
    pea       STRING2
    jsr       _toy_printString
    move.w    (sp)+,d2
    rts
STRING1:                        ; can pack them tighter
    "The sum is: "              ; down here
STRING2:
    "\n"
```

## Conclusion

In this article I have discussed the code generator for the Toy compiler, showing how it operates and what it produces. I have also discussed a number of changes that could be made to improve it. A very brief mention of some of the standard compiler optimization techniques was also given.

There are two major components of the Toy compiler left to discuss. These are the system interface code and the run-time-system. The system interface code is of interest mainly because it generates AmigaDOS object files, which are different in flavour from those of most operating systems. The run-time-system code itself is fairly simple. What is of interest here is that it is emitted directly by the compiler itself, using some fairly weird techniques. These two components will be examined in next month's article.

### Listing 1: codeGen.d - the Toy code generator

```
#toy.g
#externs.g

/*
 * codeGen.d: routines for code generation.
 */

uint CODE_BUFFER_SIZE = 10000;

/* DETAILS OF THE MC68000 CPU */

/* register limits */

uint
    D_REG_FIRST = 7,
    D_REG_LAST  = 2;

/* special register values */

uint
    R_FP        = 6,
    R_SP        = 7;

/* addressing modes and submodes */

uint
    M_DDIR      = 0o0,
    M_ADIR      = 0o1,
    M_INDIR     = 0o2,
    M_INC       = 0o3,
    M_DEC       = 0o4,
    M_DISP      = 0o5,
    M_INDEX     = 0o6,
    M_SPECIAL   = 0o7,

    M_ABS_SHORT = 0o0,
    M_ABS_LONG  = 0o1,
    M_PC_DISP   = 0o2,
    M_PC_INDEX  = 0o3,
    M_IMM       = 0o4,
    M_SR        = 0o4;

/* mode values for binary ops */

uint
    OM_REG      = 0b0 << 2,
    OM_EA       = 0b1 << 2;

/* operand size values */

uint
    S_BYTE      = 0b00,
    S_WORD      = 0b01,
    S_LONG      = 0b10,
    S_S_ADDR    = 0b011,
    S_L_ADDR    = 0b111;

/* condition code values */

uint
    CC_T        = 0x0,
    CC_F        = 0x1,
    CC_HI       = 0x2,
    CC_LS       = 0x3,
    CC_HS       = 0x4,
    CC_CC       = 0x4,
    CC_LO       = 0x5,
    CC_CS       = 0x5,
    CC_NE       = 0x6,
    CC_EQ       = 0x7,
    CC_VC       = 0x8,
    CC_VS       = 0x9,
    CC_PL       = 0xa,
    CC_MI       = 0xb,
    CC_GE       = 0xc,
    CC_LT       = 0xd,
    CC_GT       = 0xe,
    CC_LE       = 0xf;

    /* opcodes */

uint
    OP_ANDI     = 0x0200,           /* opSingle */
    OP_MOVEL    = 0x2000,           /* opMove */
    OP_MOVEW    = 0x3000,           /* opMove */
    OP_CLR      = 0x4200,           /* opSingle */
    OP_NEG      = 0x4400,           /* opSingle */
    OP_TST      = 0x4a00,           /* opSingle */
    OP_SUB      = 0x9000,           /* opModed */
    OP_CMP      = 0xb000,           /* opModed */
    OP_ADD      = 0xd000;           /* opModed */

uint
    OP_EXT      = 0x4800,           /* opSpecial */
    OP_SAVEM    = 0x48c0,           /* opSpecial */
    OP_RESTM    = 0x4cc0,           /* opSpecial */
    OP_LINK     = 0x4e50,           /* opSpecial */
    OP_UNLK     = 0x4e58,           /* opSpecial */
    OP_RTS      = 0x4e75,           /* opSpecial */
```

```
    OP_JSR      = 0x4e80,               /* opEA */
    OP_JMP      = 0x4ec0,               /* opEA */
    OP_Scc      = 0x50c0,               /* opSpecial */
    OP_Bcc      = 0x6000,               /* opBranch */
    OP_DIVS     = 0x81c0,               /* opRegister */
    OP_MULS     = 0xc1c0;               /* opRegister */

/* variables to hold the proc's code, and allocate its variables */

[CODE_BUFFER_SIZE]byte CodeBuffer;
uint CodeBufferPos;
uint NextRegister;
int GlobalVariableSize, ParameterSize, LocalVariableSize;

/* INTERFACE TO 'system.d' FOR ALLOCATION OF RUN-TIME SYSTEM VARIABLES */

/*
 * setGlobalSize - set the initial size for the global variables.
 */

proc setGlobalSize(int size)void:

    GlobalVariableSize := size;
corp;

/*
 * getGlobalSize - get the final size for the global variables.
 */

proc getGlobalSize()int:

    GlobalVariableSize
corp;

/* LOW-LEVEL ROUTINES TO PUT BYTES OF CODE INTO THE CODE BUFFER */

/*
 * genWord - generate a 16 bit word of code.
 */

proc genWord(uint w)void:

    if CodeBufferPos + 2 > CODE_BUFFER_SIZE then
        abort("code buffer overflow");
    fi;
    CodeBuffer[CodeBufferPos] := w >> 8;
    CodeBuffer[CodeBufferPos + 1] := w;
    CodeBufferPos := CodeBufferPos + 2;
corp;

/* INTERFACE ROUTINES TO BUILD PROPER MC68000 INSTRUCTIONS FOR US */

/*
 * opSpecial - generate an instruction that doesn't fit another form.
 */

proc opSpecial(uint opCode)void:

    genWord(opCode);
corp;

/*
 * opSingle - generate a single-operand instruction.
 */

proc opSingle(uint opCode, size, destEA)void:

    genWord(opCode | size << 6 | destEA);
corp;

/*
 * opEA - generate an instruction which only has an EA operand.
 */

proc opEA(uint opCode, sourceEA)void:

    genWord(opCode | sourceEA);
corp;

/*
 * opRegister - generate a register/EA type instruction.
 */

proc opRegister(uint opCode, reg, sourceEA)void:

    genWord(opCode | reg << 9 | sourceEA);
corp;

/*
 * opModed - generate an instruction with an "op-mode".
 */

proc opModed(uint opCode, reg, mode, sourceEA)void:

    genWord(opCode | reg << 9 | mode << 6 | sourceEA);
corp;

/*
 * opMove - generate a MOVE instruction.
 */

proc opMove(uint opCode, source, dest)void:

    genWord(opCode | (dest & 0o07) << 9 | (dest & 0o70) << 3 | source);
corp;

/*
 * opBranch - generate a branch instruction.
 */

proc opBranch(uint condition, offset)void:

    genWord(OP_Bcc | condition << 8 | (offset & 0xff));
corp;

/* ROUTINES CALLED FROM PARSER TO DO CODE GENERATION, ETC. */

/*
 * allocateVariable - allocate and set up a variable or parameter.
 */

proc allocateVariable(*SymbolEntry_t se)void:

    /* note: we use a size of 2 for everything. Variables of type 'bool'
       really only need 1 byte, but we use 2 to avoid any problems with
       alignment */
    case se*.se_kind
    incase sk_globalVariable:
        se*.se_value := GlobalVariableSize;
        GlobalVariableSize := GlobalVariableSize + 2;
    incase sk_parameter:
        ParameterSize := ParameterSize + 2;
        se*.se_value := -ParameterSize;
    incase sk_localVariable:
        LocalVariableSize := LocalVariableSize + 2;
        se*.se_value := -LocalVariableSize;
    esac;
corp;

/*
 * procStartParameters - set up for declaring parameters.
 */

proc procStartParameters()void:

    ParameterSize := 0;
corp;

/*
 * procEndParameters - parameters done, start with locals.
 */

proc procEndParameters()void:

    LocalVariableSize := ParameterSize + 8;
corp;

/*
 * procEndLocals - all done with locals - final offset fixup.
 */

proc procEndLocals()void:

    fixLocals(ParameterSize + 8);
    LocalVariableSize := LocalVariableSize - ParameterSize - 8;
corp;
```

```
/*
 * codeInit - initialize code generation for the current proc.
 */

proc codeInit(*SymbolEntry_t se)void:

    CodeBufferPos := 0;
    codeStartProc(se);
    opSpecial(OP_LINK | R_FP);
    genWord(- LocalVariableSize);
    opSpecial(OP_SAVEM | M_DEC << 3 | R_SP);
    genWord(0b0111111100000000);
    NextRegister := D_REG_FIRST;
corp;

/*
 * codeTerm - terminate code generation for the current proc.
 */

proc codeTerm(*SymbolEntry_t se)void:

    opSpecial(OP_RESTM | M_INC << 3 | R_SP);
    genWord(0b0000000011111110);
    opSpecial(OP_UNLK | R_FP);
    /* optimization: easier exit code if no parameters */
    if ParameterSize = 0 then
        genWord(OP_RTS);
    else
        opMove(OP_MOVEL, M_INC << 3 | R_SP, M_ADIR << 3 | 0);
        opModed(OP_ADD, R_SP, OM_REG | S_S_ADDR, M_SPECIAL << 3 | M_IMM);
        genWord(ParameterSize);
        opEA(OP_JMP, M_INDIR << 3 | 0);
    fi;
    codeEndProc(se, &CodeBuffer[0], CodeBufferPos);
corp;

/*
 * pushConstant - push a constant onto the 'stack'.
 */

proc pushConstant(int n)void:

    if NextRegister = D_REG_LAST - 1 then
        abort("out of registers");
    fi;
    /* optimization: use a CLR for the value 0 */
    if n = 0 then
        opSingle(OP_CLR, S_WORD, M_DDIR << 3 | NextRegister);
    else
        opMove(OP_MOVEW, M_SPECIAL << 3 | M_IMM, M_DDIR << 3 | NextRegister);
        genWord(n);
    fi;
    NextRegister := NextRegister - 1;
corp;

/*
 * pushVariable - push a variable onto the 'stack'.
 */

proc pushVariable(register *SymbolEntry_t se)void:

    if NextRegister = D_REG_LAST - 1 then
        abort("out of registers");
    fi;
    opMove(
        OP_MOVEW,
        if se*.se_kind = sk_globalVariable then
            M_SPECIAL << 3 | M_ABS_LONG
        else    /* sk_parameter or sk_localVariable */
            M_DISP << 3 | R_FP
        fi,
        M_DDIR << 3 | NextRegister
    );
    if se*.se_kind = sk_globalVariable then
        codeReference(rc_globalVariable, nil, CodeBufferPos);
        genWord(0);
    fi;
    genWord(se*.se_value);
    NextRegister := NextRegister - 1;
corp;

/*
 * popVariable - pop a variable from the 'stack'.
 */

proc popVariable(register *SymbolEntry_t se)void:

    NextRegister := NextRegister + 1;
    opMove(
        OP_MOVEW,
        M_DDIR << 3 | NextRegister,
        if se*.se_kind = sk_globalVariable then
            M_SPECIAL << 3 | M_ABS_LONG
        else    /* sk_parameter or sk_localVariable */
            M_DISP << 3 | R_FP
        fi
    );
    if se*.se_kind = sk_globalVariable then
        codeReference(rc_globalVariable, nil, CodeBufferPos);
        genWord(0);
    fi;
    genWord(se*.se_value);
corp;

/*
 * emitString - add a string constant to the code.
 */

proc emitString(*char st; ulong length)void:
    register *char p;
    register ulong l;

    /* round the length up to an even value. We are cheating ever so slightly
       here in that we will reference the byte in memory beyond the allocated
       length of the string. We know, however, that any memory allocator that
       we are likely to encounter will round requests up to at least an even
       number of bytes, so we should be OK */
    l := length;
    if l & 1 ~= 0 then
        l := l + 1;
    fi;
    /* we do a JSR around the string, which results in its address being
       on the stack - just where we want it to call 'toy_printString' */
    opEA(OP_JSR, M_SPECIAL << 3 | M_PC_DISP);
    genWord(l * sizeof(char) + 2);
    p := st;
    while l ~= 0 do
        l := l - 1;
        if CodeBufferPos = CODE_BUFFER_SIZE then
            abort("code buffer overflow");
        fi;
        CodeBuffer[CodeBufferPos] := p* - '\e';
        CodeBufferPos := CodeBufferPos + 1;
        p := p + sizeof(char);
    od;
    memFree(st, length * sizeof(char));
corp;

/*
 * callSpecial - call one of the special run-time routines.
 */

proc callSpecial(RelocCode_t rc)void:

    if rc = rc_printInt then
        NextRegister := NextRegister + 1;
        opMove(OP_MOVEW, M_DDIR << 3 | NextRegister, M_DEC << 3 | R_SP);
    fi;
    opEA(OP_JSR, M_SPECIAL << 3 | M_ABS_LONG);
    codeReference(rc, nil, CodeBufferPos);
    genWord(0);
    genWord(0);
    if rc = rc_readInt then
        if NextRegister = D_REG_LAST - 1 then
            abort("out of registers");
        fi;
        opMove(OP_MOVEW, M_DDIR << 3 | 0, M_DDIR << 3 | NextRegister);
        NextRegister := NextRegister - 1;
    fi;
corp;

/*
 * pushParameter - push a parameter to a proc call.
 */
```

```
proc pushParameter()void:

    NextRegister := NextRegister + 1;
    opMove(OP_MOVEW, M_DDIR << 3 | NextRegister, M_DEC << 3 | R_SP);
corp;

/*
 * callProc - generate a call to the given procedure or function.
 */

proc callProc(*SymbolEntry_t se)void:

    opEA(OP_JSR, M_SPECIAL << 3 | M_ABS_LONG);
    if se*.se_kind = sk_procedure then
        codeReference(rc_none, se, CodeBufferPos);
        genWord(0);
        genWord(0);
    fi;
    if se*.se_type ~= rt_void then
        if NextRegister = D_REG_LAST - 1 then
            abort("out of registers");
        fi;
        opMove(OP_MOVEW, M_DDIR << 3 | 0, M_DDIR << 3 | NextRegister);
        NextRegister := NextRegister - 1;
    fi;
corp;

/*
 * procResult - put a proc's result in the proper place (D0).
 */

proc procResult()void:

    NextRegister := NextRegister + 1;
    opMove(OP_MOVEW, M_DDIR << 3 | NextRegister, M_DDIR << 3 | 0);
corp;

/*
 * unaryOp - perform a unary operation on the current 'top-of-stack'.
 *      (only unary operation currently is negate)
 */

proc unaryOp(TokenKind_t opToken)void:

    opSingle(OP_NEG, S_WORD, M_DDIR << 3 | (NextRegister + 1));
corp;

/*
 * binaryOp - perform a binary operation on the current 'top-of-stack'.
 */

proc binaryOp(TokenKind_t opToken)void:

    NextRegister := NextRegister + 1;
    if opToken = tk_star or opToken = tk_slash then
        if opToken = tk_slash then
            opSpecial(OP_EXT | 0b011 << 6 | (NextRegister + 1));
        fi;
        opRegister(
            if opToken = tk_star then
                OP_MULS
            else
                OP_DIVS
            fi,
            NextRegister + 1,
            M_DDIR << 3 | NextRegister
        );
    else
        opModed(
            if opToken = tk_plus then
                OP_ADD
            else
                OP_SUB
            fi,
            NextRegister + 1,
            OM_REG | S_WORD,
            M_DDIR << 3 | NextRegister
        );
    fi;
corp;

/*
 * compareValues - compare two values, setting result accordingly.
 */

proc compareValues(TokenKind_t opToken)void:

    NextRegister := NextRegister + 1;
    opModed(OP_CMP, NextRegister + 1, OM_REG | S_WORD,
            M_DDIR << 3 | NextRegister);
    opSpecial(OP_Scc |
                case opToken
                incase tk_equal:
                    CC_EQ
                incase tk_notEqual:
                    CC_NE
                incase tk_less:
                    CC_LT
                incase tk_lessEqual:
                    CC_LS
                incase tk_greater:
                    CC_GT
                incase tk_greaterEqual:
                    CC_GE
                esac << 8 |
                M_DDIR << 3 | (NextRegister + 1)
    );
    opSingle(OP_ANDI, S_WORD, M_DDIR << 3 | (NextRegister + 1));
    genWord(1);
corp;

/*
 * getPos - return the current code position, for later backpatching.
 */

proc getPos()uint:

    CodeBufferPos
corp;

/*
 * branchForward - branch to some as-yet-unknown forward location. Return
 *      the position to be used later to fill in the offset.
 */

proc branchForward(bool isConditional)uint:

    if isConditional then
        NextRegister := NextRegister + 1;
        opSingle(OP_TST, S_WORD, M_DDIR << 3 | NextRegister);
        opBranch(CC_EQ, 0);
    else
        opBranch(CC_T, 0);
    fi;
    genWord(0);
    CodeBufferPos - 2
corp;

/*
 * patchBranch - patch a previous branch to come to the current position.
 */

proc patchBranch(register uint offset)void:

    CodeBuffer[offset] := (CodeBufferPos - offset) >> 8;
    CodeBuffer[offset + 1] := (CodeBufferPos - offset);
corp;

/*
 * branchTo - generate a backwards unconditional branch.
 */

proc branchTo(uint offset)void:
    register uint distance;

    distance := CodeBufferPos + 2 - offset;
    /* optimization: use a short branch if possible */
    if distance <= 128 then
        opBranch(CC_T, - distance);
    else
        opBranch(CC_T, 0);
        genWord(- distance);
    fi;
corp;
```

HOURS:
8 AM to 7 PM Monday - Friday
8 AM to 4 PM Saturday
Call on Sunday

# The Software Shop, Inc.

## Your AMIGA Software and Hardware Marketplace

Visa and MasterCard Accepted

# Hardware Spectacular! Sensational Savings! Compare!

| HardCards w/HF2000 | | HardCards w/FastCard | | FastTrack A500/A1000 | | ToolBox A500/A1000 | | Memory Expansion A2000/A2500/A500/A1000 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HardFrame | $ 249 | FastCard | $ 159 | FastTrack Sys. | $ 379 | ToolBox | $ 249 | 8Up 2Mg | $ 489 | Starboard Ok | $ 269 |
| 138N 32Mg | 589 | 138N 32Mg | 506 | 138N 32Mg | 722 | ToolBox w/48Mg | 849 | 8Up 4Mg | 769 | Starboard 512K | 359 |
| 157N 48Mg | 669 | 157N 48Mg | 589 | 157N 48Mg | 799 | ToolBox w/32Mg | 769 | 8Up 6Mg | 1,059 | Starboard 1Mg | 449 |
| Quantum 40Mg | 759 | Quantum 40Mg | 689 | Quantum 40Mg | 899 | ToolBox w/Q40 | 939 | 8Up 8Mg | 1,349 | Starboard 2Mg | 669 |
| Quantum 80Mg | 1,089 | Quantum 80Mg | 1,039 | Quantum 80Mg | 1,223 | ToolBox w/Q80 | 1,279 | Spirit 512K | 319 | MiniMeg 512K | 378 |
| Quantum 105Mg | 1,179 | Quantum 105Mg | 1,099 | Quantum 105Mg | 1,304 | ToolBox w/Q105 | 1,349 | Spirit 1Mg | 411 | MiniMeg 1Mg | 499 |
| 277N 63Mg | 759 | 277N 63Mg | 689 | 277N 63Mg | 889 | ToolBox w/63Mg | 939 | Spirit 105Mg | 499 | MiniMeg 2Mg | 735 |

External 5.25 Drv. $179 External 3.5 Drv. $119 Internal 3.5 Drv. $89 Digi-View Gold, WV1410 Panasonic camera w/lens and Copy stand $399

### ACCESSORIES

| | |
|---|---|
| A-B Switch cable | $35.00 |
| A/B/C/D Switchbox | 63.00 |
| Boing optical mouse | 109.00 |
| Copy Stand | 65.00 |
| Ece Midi 1000 | 52.00 |
| Ece Midi 500/2000 | 52.00 |
| Epyx 500JX | 15.00 |
| Ergo Joystick | 17.00 |
| Freedom stick | 65.00 |
| Infared mouse | 176.00 |
| MD120 3.5 Disk holder | 29.00 |
| MD60 disk holder | 26.00 |
| MD64 disk holder | 26.00 |
| Modem cable A1000 | 15.00 |
| Modem cable A2000/500 | 15.00 |
| Mouse holder | 6.00 |
| Mouse master | 29.00 |
| Mouse pad | 9.00 |
| Multisync cable I-102M/F | 24.00 |
| Okimate black ribbon | 5.00 |
| Okimate color ribbon | 8.00 |
| Power strip 6 outlets | 17.00 |
| Printer cable 2000/500 | 15.00 |
| Printer cable A1000 | 15.00 |
| Quick shot Joystic | 15.00 |
| Ribbon panasonic all | 10.00 |
| Slick stick | 8.00 |
| Star NX black ribbon | 8.00 |
| Star NX color ribbon | 12.00 |
| Tilt and swivel stand | 21.00 |
| Xerox 4020 500/sheet | 25.00 |

### CAD SYSTEM

| | |
|---|---|
| Draw 2000 | $183.00 |
| Home builder cad | 129.00 |
| Home builder choice | 53.00 |
| Intro cad | 51.00 |
| Logic works | 65.00 |
| Pro-Board | 399.00 |
| Pro-Net | 399.00 |
| X-Cad system | 385.00 |

### A500/A2000/A2500 PERIPHERALS

| | |
|---|---|
| A1084S stereo monitor | CALL |
| A2000 HD 40mg28ms w/cont | CALL |
| A2058 2mg 8mg ram board | CALL |
| A20620/2meg 32bit | CALL |
| A20620/4mg 32bit Mem | CALL |
| A2090A HD controller | CALL |
| A2300 Genlock | CALL |
| A2500 HD40mg28ms3mgMem 68020/68881 | CALL |
| Amiga 2000 Cpu | CALL |
| Amiga 500 CPU | CALL |
| AT Bridge card A2286 | CALL |
| Bridge card A2000 | CALL |

### COMMUNICATIONS

| | |
|---|---|
| Atalk III | $65.00 |
| Atredes profess.BBS | 96.00 |
| BBS pc | 96.00 |
| Diga | 50.00 |
| Online Platinum | 69.00 |

### CREATIVE PRODUCTIVITY

| | |
|---|---|
| Analytic art | $39.00 |
| Gizmoz enhanced | 45.00 |
| Intellitype | 35.00 |
| Magellan | 129.00 |
| Master type | 25.00 |
| Mavis beacon typing | 33.00 |
| Your family tree | 32.00 |

### DATABASE MANAGEMENT

| | |
|---|---|
| Acquisition | $189.00 |
| Data retrieve | 51.00 |
| Data Retrieve Prof | 189.00 |
| Dbman | 125.00 |
| Micro base | 24.00 |
| Micro fiche filer | 64.00 |
| Microfiche filer plus | 115.00 |
| Organize | 49.00 |
| Softwood file II gs | 79.00 |
| Super base | 52.00 |
| Superbase pers. II | 98.00 |
| Superbase Pro 3.0 | 229.00 |
| Superbase professional | 189.00 |

### PROGRAMMING LANGUAGES

| | |
|---|---|
| A/C basic | $129.00 |
| A/C fortran | 195.00 |
| Algebra ii | 33.00 |
| Arexx | 33.00 |
| Assem pro | 65.00 |
| Aztec C developer | 195.00 |
| Aztec C professional | 129.00 |
| Aztic complete devel.sys. | 315.00 |
| Benchmark C library | 60.00 |
| Benchmark iff library | 62.00 |
| Benchmark modula 2 | 128.00 |
| Benchmark simplified | 62.00 |
| Cape 68k | 59.00 |
| Critic's Choice | 159.00 |
| Devpac Ed/Assem/link/debg | 66.00 |
| Dissasembler | 45.00 |
| GFA basic 3.0 | 95.00 |
| Inovatools #1 | 54.00 |
| Lattice C | 132.00 |
| Lattice C + + | 379.00 |
| Lattice C professional | 249.00 |
| Lattice developemnt system 5.0 | 199.00 |
| Library's source | 197.00 |
| Macro assembler | 65.00 |
| Metacomco lisp | 131.00 |
| Metacomco pascal | 95.00 |
| Metacomco shell | 40.00 |
| Metascope debugger | 65.00 |
| Power windows v2.5 | 58.00 |
| Pro Basic | 124.00 |
| Publishers choice | 130.00 |
| Source level debugger | 50.00 |
| The power system pecan | 93.00 |
| The professional pecan | 89.00 |
| The Works Platinum | 164.00 |
| True basic | 65.00 |
| W. shell | 33.00 |

### FONTS

| | |
|---|---|
| Asha's fonts | 57.00 |
| Calligrapher | $63.00 |
| Fancy 3d fonts | 52.00 |
| Font set I | 22.00 |
| Inter font | 76.00 |
| Kara fonts color | 50.00 |
| Lion fonts | 56.00 |
| Masterpiece fonts | 159.00 |
| News letter fonts | 29.00 |
| Profonts I Prof | 23.00 |
| Profonts II decorative | 23.00 |
| Studio font | 29.00 |

### HARDWARES

| | |
|---|---|
| 8Up Memory Exp. A2000 | $165.00 |
| Accelerator | 152.00 |
| Amigen | 153.00 |
| Easyl A1000 | 349.00 |
| Easyl A2000 | 379.00 |
| Easyl A500 | 345.00 |
| Expan 512K A500 | 159.00 |
| Expansion 1mg A500 | 409.00 |
| Flicker fixer | 495.00 |
| Framegrabber | 579.00 |
| Future sound | 129.00 |
| Light pen | 99.00 |
| Live A2000 | 349.00 |
| Live framegrabber A1000 | 245.00 |
| Live framegrabber A500 | 305.00 |
| MC68881 Math Co-processor | 179.00 |
| Multifunction card | 75.00 |
| My-T-Mouse | 39.00 |
| Perfect sound A1000 | 69.00 |
| Perfect sound A500/A2000 | 69.00 |
| Phoenix power supply A500 | 95.00 |
| Progen | 379.00 |
| Super Gen | 695.00 |
| Supra modem 2400bd | 149.00 |
| WV1410 Panasonic Camera w/lens | 199.00 |

### SPREADSHEETS

| | |
|---|---|
| Analyze! 2.0 | $95.00 |
| Haicalc | 32.00 |
| Maxiplan 500 | 95.00 |
| Maxiplan plus | 126.00 |
| Super plan | 97.00 |
| VIP Professional | 65.00 |

### DESKTOP PUBLISHING

| | |
|---|---|
| Citydesk | $97.00 |
| Citydesk companion | 15.00 |
| Citydesk companion II | 15.00 |
| Page stream | 129.00 |
| Page setter | 97.00 |
| Professional Draw | 129.00 |
| Professional page | 249.00 |
| Publisher plus | 69.00 |
| Shakespeare | 129.00 |

### ACCOUNTING

| | |
|---|---|
| The Accountant | $183.00 |
| Best general ledger | 77.00 |
| Financial plus | 189.00 |
| Invester advantage | 63.00 |
| Miamiga ledger | 63.00 |
| Micro lawyer | 39.00 |
| Easy Ledger | 199.00 |

### WORDPROCESSORS

| | |
|---|---|
| Becker text | $92.00 |
| Excellence! | 179.00 |
| Flow idea processors | 63.00 |
| Gold spell II | 30.00 |
| Kindwords | 65.00 |
| LPD writer | 75.00 |
| LPD writer unprotected | 83.00 |
| Micro text | 22.00 |
| Prowrite v2.0 | 80.00 |
| Reason | 249.00 |
| Text pro | 50.00 |
| Word perfect | 159.00 |
| Wp library | 79.00 |

### MONITORS

| | |
|---|---|
| Amdek multisync 12' mon. | $512.00 |
| Panasonic 8cm515 mon. | 249.00 |
| Sieko cm1430 monitor | 699.00 |
| Sony multi-scan Monitor | 535.00 |
| Zenith 14' Flat screen mon. | 720.00 |
| Nec 3D multisync | 799.00 |
| Taxan 1000 20" multisync | 2899.00 |

### EDUCATION

| | |
|---|---|
| Adventure of sinbad | 32.00 |
| Aesop's fables | 32.00 |
| All about America | 35.00 |
| Animal kingdom | 32.00 |
| Decimal dungeon | 32.00 |
| Discovery game math | 25.00 |
| Discovery game spell | 25.00 |
| First letters and words | 35.00 |
| First shapes | 35.00 |
| Fraction action | 32.00 |
| Grade manager | 59.00 |
| Great states | 25.00 |
| Kid Talk | 35.00 |
| Kinderama | 32.00 |
| Kwik speak spanish | 19.00 |
| Learning curve | 52.00 |
| Link word French | 22.00 |
| Link word German | 22.00 |
| Link word Italian | 22.00 |
| Link word Russian | 22.00 |
| Link word Spanish | 22.00 |
| Math talk | 35.00 |
| Math talk fraction | 35.00 |
| Math wizard | 35.00 |
| Perfect score | 51.00 |
| Quiz master | 51.00 |
| Read & rhyme | 32.00 |
| Read-a-rama | 32.00 |
| Spell bound | 25.00 |
| Speller bee | 35.00 |
| Tales from Arabia | 32.00 |
| Wordmaster | 32.00 |

### PRINTERS

| | |
|---|---|
| 1091i Panasonic printer | $224.00 |
| Alps Allegro 24 PIN | 399.00 |
| Epson Apex 80 | 205.00 |
| NEC LC890 laser PS. | 3395.00 |
| Okimate 20 color prt | 145.00 |
| Plug'n print | 85.00 |
| Star nx rainbow | 259.00 |
| Xerox 4020 ink jet | 1295.00 |
| Xerox 4020 starter kit | 159.00 |
| HP Paint jet | CALL |

### SOUND & MUSICS

| | |
|---|---|
| 4-op deluxe | $97.00 |
| Audio master | 37.00 |
| Audio master II | 67.00 |
| Back songbook | 27.00 |
| Copyist II | 179.00 |
| D-50 | 99.00 |
| Deluxe music | 69.00 |
| Dr drums | 20.00 |
| Dr keys | 20.00 |
| Dr T's Keyboard | 160.00 |
| Dr. T's Midi recording studio | 45.00 |
| Dx heaver | 97.00 |
| Dynamic drums | 52.00 |
| Dynamic studio | 129.00 |
| Future sound A500/A2000 | 92.00 |
| Hot cool jazz | 18.00 |
| Hot licks | 32.00 |
| Instant music | 32.00 |
| Matrix 6 | 97.00 |
| Midi Magic | 99.00 |
| Mt-32 | 97.00 |
| Music student | 38.00 |
| Pro sound designer | 125.00 |
| Promidi studio | 129.00 |
| Rock & roll | 21.00 |
| Sonix | 51.00 |
| Sound sampler | 78.00 |
| Studio magic | 65.00 |
| Synthia | 59.00 |
| Texture | 97.00 |
| Utilities 2(mimatics) | 43.00 |

### UTILITIES

| | |
|---|---|
| B.A.D.disk optimizer | $32.00 |
| C.B. Tree | 65.00 |
| Celebrity cook book | 20.00 |
| Custom screens * | 45.00 |
| Cygnused Professional | 65.00 |
| Disk 2 disk | 33.00 |
| Disk master | 33.00 |
| Disk mechanic | 59.00 |
| Dos 1.3 | 22.00 |
| Dos 2 dos | 35.00 |
| Doug's math aquarium | 52.00 |
| Encore | 45.00 |
| Ez backup | 33.00 |
| Facc II | 22.00 |
| Fine print | 33.00 |
| Fleet check | 33.00 |
| Flip side | 37.00 |
| Gold spell | 32.00 |
| Gomf | 27.00 |
| Grabbit | 19.00 |
| Laser scripts | 28.00 |
| Laser up: fonts | 26.00 |
| Laser up: plot | 33.00 |
| Laser up: print | 58.00 |
| Laser up: utilities | 26.00 |
| Lex check | 33.00 |
| Lv backup | 32.00 |
| Multi preferences | 20.00 |
| Nag Plus Schedule Assist. | 52.00 |
| Pro script | 32.00 |
| Project D | 32.00 |
| Promis spelling | 32.00 |
| Quarterback | 45.00 |
| Raw copy 1.3 | 39.00 |
| Superback | 52.00 |
| Sys font * | 22.00 |
| Tax plan | 22.00 |
| Tdi editor sources | 30.00 |
| Tdi examples | 15.00 |
| Tdi grid access | 30.00 |
| Tdi kermit | 19.00 |
| Text ed plus | 52.00 |
| V.I.P. | 32.00 |

### HOME ACCOUNTING

| | |
|---|---|
| Desktop budget | $46.00 |
| Money mentor | $65.00 |
| Phaser | 65.00 |
| Tax Break | 53.00 |

### GRAPHICS & VIDEO

| | |
|---|---|
| 3-Demon | $69.00 |
| Animate 3-D | 99.00 |
| Animate 4D | 439.00 |
| Animation editor | 39.00 |
| Animation effects | 32.00 |
| Animation Flipper | 32.00 |
| Animation multiplane | 58.00 |
| Animation stand | 32.00 |
| Animation w/images | 89.00 |
| Animator apprentices | 185.00 |
| Animotion | 65.00 |
| Architechtural design | 23.00 |
| Art gallery I | 20.00 |
| Art gallery II | 20.00 |
| Art pak I | 20.00 |
| Art parts 1 | 20.00 |
| Art parts II | 20.00 |
| Award maker plus | 32.00 |
| Broadcast Titler | 189.00 |
| Butcher | 25.00 |
| C light 3d editor | 39.00 |
| Calligrapher help | 28.00 |
| Chroma Paint | 48.00 |
| Color splitters | 79.00 |
| Comicsetter | 65.00 |
| Deluxe paint II | 89.00 |
| Deluxe Paint III | 102.00 |
| Deluxe print II | 55.00 |
| Deluxe photo lab | 102.00 |
| Deluxe production | 137.00 |
| Deluxe video II | 89.00 |
| Designasaurus | 32.00 |
| Digi paint | 39.00 |
| Digi paint help | 24.00 |
| Digi view | 131.00 |
| Director's Toolkit | 26.00 |
| Dpaint help | 24.00 |
| Exprss paint | 65.00 |
| Express paint III | 89.00 |
| Forms in flight II | 77.00 |
| Future design 3-d | 23.00 |
| Graphic studio * | 35.00 |
| Human design 3-d | 23.00 |
| Icon Paint | 59.00 |
| Impact | 55.00 |
| Interchange | 32.00 |
| Invision | 109.00 |
| Lights Camera Action | 49.00 |
| Microbot design 3-d | 23.00 |
| Modeler 3d | 64.00 |
| Movie setter | 65.00 |
| Page flipper fx | 95.00 |
| Perfect vision | 175.00 |
| Photon expansion disk | 19.00 |
| Photon paint | 65.00 |
| Photon paint 2.0 | 97.00 |
| Photon paint help | 23.00 |
| Photon video cell anim. | 97.00 |
| Pixmate | 43.00 |
| Print master plus | 33.00 |
| Prism | 43.00 |
| Pro video plus set I | 83.00 |
| Pro video plus set II | 83.00 |
| Pro. result using dpaint | 35.00 |
| Provideo plus | 193.00 |
| Sculpt 3-d | 64.00 |
| Silver turbo | 129.00 |
| The Director | 46.00 |
| Turbo silver | 129.00 |
| Tv-show | 64.00 |
| Tv-text | 64.00 |
| Video effects 3d | 121.00 |
| Video scape 3-d | 120.00 |
| Video titler | 95.00 |
| Zoetrope | 91.00 |

### DISKETTES

| | |
|---|---|
| Fuji 3.5' ds/dd color disks | $19.00 |
| Sony 3.5' ds/dd | 19.00 |
| TDK 3.5' ds/dd disks | 19.00 |
| Centech 3.5' ds/dd disks | 19.00 |
| Centech 5.25' ds/hd disks | 29.00 |

# 1-800-752-0050

In Mass. Call....

# 1-508-756-6452

**SALES AND QUOTES (outside Mass.):**
**1-800-752-0050**

**SHIPPING AND TECHNICAL SERVICE:**
**1-508-756-6452**

**QUOTE-A-FAX: 1-508-799-9354**

- Corporate and volume purchases invited. Call Moe.
- Stock orders placed by 12:00 NOON EST shipped the same day.
- Allow 10 days for personal checks to clear.
- Wire transfer accepted.
- Minimum $5.00 Shipping/handling charge
- Add $3.50 for COD.
- Returns subject to restocking charge
- Software defects-call the Manufacturer
- For faster delivery send cashiers check

### ENTERTAINMENT SOFTWARE

***We have them all! Over 300 games in stock. Call for your favorite game, and check our low prices.***

**SHOWROOM: 22 Front Street, Worcester, MA 01614**

# Introduction to Usenet

## *A survival guide for the novice*

**by Tim Grantham**
*Editor*

The UNIX operating system (OS) has rapidly emerged from the halls of academia into the mainstream of computing during the last ten years. There are versions running on everything from Cray supercomputers to PCs – and on the Amiga as AMIX. There are many, myself included, who believe UNIX will be *the* standard operating system in five years. That doesn't mean that AmigaDOS, MS-DOS or any other major OS will fade away into oblivion; they all have particular features that UNIX cannot match, not to mention their vast application software base.

But UNIX and its descendants will become the standard for one very basic reason: it's a *portable* operating system. Whenever a new type of microprocessor appears nowadays, be it the Motorola 88000, the Sun SPARC chips or the MIPS R3000, UNIX is always the first operating system available. While no operating system can provide what every user wants, UNIX provides the basics: multitasking, multiple users, virtual memory, programming tools and many others.. There are even versions of UNIX for parallel processing systems: the latest workstations from Silicon Graphics, for example, use one, two, four or eight MIPS R3000 RISC-type microprocessors running IRIX, a fully-threaded version of UNIX that provides transparent multiprocessing.

It may be well worth your while, then, to begin to familiarize yourself with UNIX. There are many books on the subject: one that I can recommend is *UNIX: The Complete Reference, System V, Release 3*, by Stephen Coffin, published by Osborne McGraw-Hill (reviewed in Volume 2, Issue 3, p. 68). As you delve further into the system, you may also find the series of books published by O'Reilly And Associates very valuable. These are available in many college bookstores, or directly from the publisher at 632 Petaluma Ave. Sebastopol, California, 95472. The company can also be contacted by telephone at (800) 533-NUTS (in the continental US only).

By far the most enjoyable way, in my opinion, to begin your life with UNIX, is to become a reader of the Amiga newsgroups on Usenet. The signal-to-noise content of the newsgroups is generally very high; and many of the Amiga world's brightest lights post articles and programs on a regular basis, including Matt Dillon, Larry Phillips, Chuck MacManis, Dave Haynie, Randall Jesup and others.

### What is Usenet?

What follows is a necessarily simplified explanation of Usenet; UNIX and networking is an enormously complex subject.

Usenet is not something that exists on a few dedicated machines, like CompuServe, PeopleLink or BIX. Rather, it's a global network of computer systems, linked predominantly by telephone lines, but also by Ethernet and other transmission media.

Distributing news and mail is not the primary function of these computers; they're usually being used as full-fledged systems for every possible application: business, technical and arts. But at least once every 24 hours (usually at night), a UNIX system will connect with another UNIX system (its 'feed' system) and download the latest news, mail and programs into its *spool* directory, using the program *uucp* (UNIX to UNIX copy). From there, the material is made available to users on the system; or, if it consists of mail addressed to users on other computers, it is held for transmission in turn to the systems that this particular node feeds. It is possible for a message to have wended its way through many machines before it arrives at its destination. In this respect, Usenet is similar to FidoNet and PunterNet. Mail and news can be days old by the time it reaches your machine, or only a few seconds, depending on the type of transmission used for each leg of the journey.

### Using Usenet

The most direct way to access Usenet is by obtaining an account on a UNIX system with a newsfeed. Most universities and colleges now have at least one UNIX machine and the necessary communications hardware. They may make accounts available to the general public, sometimes for a fee. You can also access Usenet through FidoNet, PNet, The Portal System, Canada Remote Systems, and other large bulletin board systems. Alternatively, more and more businesses are buying UNIX systems; you may be able to get an account at the place you work.

Finally, of course, you can buy your own UNIX system. The more expensive ones, like the Personal Iris from Silicon Graphics, have very nice windowing user interfaces and simplified system administration software that make things easier for the UNIX novitiate.

The most common way to access a UNIX account, however, is via a terminal, usually running on a personal computer connected by modem or local area network. I use Eric Haberfellner's Handshake VT100 terminal emulation program on my A1000; my account is on an AT&T 3B1 UNIX machine owned and operated by Chris Siebenmann.

Once the system administrator has established your account, you can log on and begin reading the news and sending and receiving mail. The program I use for the latter is *rn*; for mail, I use *mhmail*. Whatever the corresponding programs are on your host, be it *vn* or *mailx*, you can usually obtain online help with the *man* program, i.e. **man rn** or **man mailx**.

You will also need to become familiar with one of the system editors, in order to write messages or mail: *vi* or *emacs* are probably the two most common. Both are very powerful, though neither is friendly to strangers.

The *rn* program provides a highly-configurable means of reading news. I invoke the program like this:

```
rn -e -m -S -/ comp.sys.amiga.
```

This lets me, among other things, read the news in the newsgroup comp.sys.amiga in subject-threaded order, pausing at the end of each screenful of data. The *rn* program gives me an opportunity to respond publicly to each message, or to send email directly to the author. The *rn* program also lets you save programs posted to comp.binaries.amiga but you will need the programs *uudecode* and *zoo* to unscramble and uncompress them.

The various newsgroups on Usenet are an incredibly valuable resource. Many of the top minds in the world contribute on a regular basis. And not just on computers: there is a newsgroup for almost any subject you can think of, from rec.arts.poetry to alt.sex.bondage.

The following pages contain a very useful compendium of Usenet basics, and answers to common questions compiled and posted on a semi-regular basis by Edwin Hoogerbeets at the University of Waterloo. Read it before attempting to post news and keep it by your side when you're logged on. I'm sure I'm not alone in thanking Edwin for taking the time to put this information together.

```
Article 17224 of comp.sys.amiga:
From: ehoogerbeets@rose.waterloo.edu (Edwin Hoogerbeets)
Newsgroups: comp.sys.amiga
Subject: Introduction to comp.sys.amiga
Message-ID: <14872@watdragon.waterloo.edu>
Date: 4 Jul 89 19:22:32 GMT
Expires: Aug  4 15:19:13 EDT 1989
Sender: daemon@watdragon.waterloo.edu
Reply-To: ehoogerbeets@rose.waterloo.edu (Edwin Hoogerbeets)
Organization: U. of Waterloo, Ontario
Lines: 1539


This is an introductory posting to comp.sys.amiga. If you are a new
reader of this group or even if you just want Amiga/Usenet information,
please read the following articles.

This posting last changed:  July 4, 1989

It is recommended that you save this article for future reference.
Please refer to this article first before posting to the net.  This
will help keep the comp.sys.amiga* groups uncluttered with topics that
have already been discussed.

*
* There is an analogous posting in comp.sys.amiga.tech.
*

To save this posting when you are using
rn or vn: s newuser<return> to save in the file News/newuser.
notes: snewuser<return> to save this article in the file ./newuser.

To read any of the following topics after this page, type 'g' for 'go
to' and the three letter index identifier in capitals that is listed on
the left. This article contains the following topics:

Please note that the list of FTP sites has been deleted from this
posting, since Tuna Ertemalp posts the same list monthly. Also, since
he is the one taking care of the list, he will always have the more
up-to-date version.

,NET   The Etiquette of the net. (netiquette) Please read this article
       before posting anything.
         edwin@hcr.UUCP
,MOD   How to receive and post Amiga programs from/to the moderated Amiga
       newsgroups.
         page@Sun.COM (Bob Page)
         ain@mace.cc.purdue.edu
,FNF   The Fish Public Domain software library. How to get Fish disks
       and how to submit programs.
         fnf@fishpond.UUCP
         ali@polya.stanford.edu
,PIC   How to get and send Amiga graphic pictures and related 'show'
       programs.
         joe@dayton.UUCP
,ARC   How to use the kilowatt archive server
                                raz@sun.COM (Steve -Raz- Berry)
,WED   Address of the Wedge people.
         stan@teroach.UUCP
,SID   Putting the Amiga 2000 on its side. Is it possible?
         edwin@hcr.UUCP
,BUY   Which machine to buy? (500 or 2000 Amiga, Atari, Mac?)
         uunet!Sun.COM!cmcmanis (Chuck McManis)
         geoffs@smoke.UUCP
         AmigaIan@cup.portal.com.UUCP
,CPU   Is 14.32MHz too fast for my expansion RAM?
         daveh@cbmvax.UUCP
,LED   What does flashing caps lock LED mean?
         bjc@pollux.UUCP
,MAC   A2000 vs B2000, which machine is which?
         daveh@cbmvax.UUCP
,SUR   comp.sys.amiga survey, opinions from the net
         uunet!rice.edu!carson
,CLI   Closing the initial CLI
         hcr!edwin
,KIK   Crashing all the way to Kickstart 1.3 (*the solution*)
         bryce@cbmvax
,HAR   Hard drives for Amiga (especially the 2000)
         Witting@topaz.rutgers.edu
,COL   All dem [screen] colors during boot
         hah@mipon3.intel.com (Hans Hansen)
,GUR   GURU-NUMBERS (what they mean)
         bryce@hoser.berkeley.edu (Bryce Nesbitt)
         (current address: bryce@cbmvax)
,HED   The 2090(a) and SCSI drives
         jesup@cbmvax

Send questions, comments to:

Edwin
ehoogerbeets@rose.waterloo.edu
*************************************************************************
```

,NET
From: edwin@hcr.UUCP (Edwin Hoogerbeets)
Subject: Netiquette

Behaviour on the net:
Behaviour on the net should be based on respect for other readers. If you follow this guideline, the net can be a very powerful tool for useful information.

Try to reduce the amount of material posted and keep your articles short. This helps to save on phone bills throughout the entire Usenet community and saves time when reading the voluminous Amiga news groups.

Try to avoid posting your view on an issue that has no "correct" answer. An example of this is the "my machine is better than your machine" discussion that often flares up in the microcomputer newsgroups. If you must express your point of view, please indicate that you would like to continue the discussion via mail, so as not to harass other readers that are not interested in the subject.

Please do not make derogatory postings (aka. flames). Do not subject other readers to these childish tit for tat postings. Have fun via mail.

Newsgroups:
There are currently four official groups dedicated to the Amiga. Here is a brief description of each of them and what kind of articles they should contain:

comp.sys.amiga - This newsgroup is for general discussion of Amiga related news, software and hardware.

comp.sys.amiga.tech - This newsgroup is for the technical discussions about programming the hardware and software of the Amiga.

comp.sources.amiga - This newsgroup contains the sources to freely redistributable programs posted exclusively by the moderators.

comp.binaries.amiga - This newsgroup contains the binaries related to the programs in the sources group. This group can also only be posted to by the moderators.

Please adhere to these classifications and post your news to the appropriate group. See the MOD section (enter g MOD) in this posting on how to post to and receive from the moderated groups.

Mail:
If you would like to reply to an article posted by someone else and you do not think your reply is of value to others on the net, then please send mail to the poster instead of posting a reply. This helps reduce the amount of news sent each day and improves the ratio of signal to noise in the newsgroup. When in rn type 'r' to reply via mail. Type 'R' to include a copy of the poster's article.

Following up:
If you do feel the reply is important enough for everyone on the net, then please keep your posting small. In rn, type 'f' to follow up, and 'F' to include a copy of the poster's original article. Please trim the included article down to only the relevant parts, so that people are reminded of the gist of the original without having to re-read the entire text.

Posting original material:
Original material is preferred, as it offers something new for the readers. However, if you post an request for information or software, please include a request for mail instead of a news followup. Again, this is to help reduce the amount the reader must go through each day.

Cross posting:
Cross posting to many newsgroups takes no more room or time than posting to one newsgroup. However, many more people see your article, and the above rules become more critical. If you do cross post, please realize that people from any of the groups may followup to your article. By default, most news programs post to all the original newsgroups. If you think the replies to your article will only be relevant to one group, please use the Followup-To line to indicate that the named news group should be used for replies. Also, if you are replying to someone else's cross-posted article, please edit the Newsgroups line to indicate only the groups to which your reply is relevant.

Remember to have respect for your fellow netters and you will gain the most from Usenet.

Edwin
uunet!utai!utcsri!hcr!edwin
**********
,MOD
From: page@Sun.COM (Bob Page)
Subject: how to retrieve programs from the moderated groups

How To Use The Comp.Binaries.Amiga Postings

Every comp.binaries.amiga submission is archived with the ZOO archive program, which does a fair amount of data compression and retains the original names, dates and directory structure within the archive.

Because Usenet data paths are not 8-bit, these archives must be encoded to "regular ASCII" which is 7-bit. The encoding is done with a program called 'uuencode'. The version used adds a checksum at the end of each line, and includes the file size at the end of the encoding.

Sometimes the files are too large for one posting (Usenet postings should generally be less than 64k bytes but you can sometimes get away with longer postings). If this uuencoded zoo archive is too large, it gets split into pieces, roughly 64k in length.

Finally, the (possibly split) .zuu files are enclosed by some simple UNIX commands. This 'shell archive' is commonly known as a 'shar'.

There is a more-or-less standard set of file extensions used to show the file type. The file extension is the set of characters after the dot (or period) in the name. The set of file extensions used is:

| | |
|---|---|
| .zoo | An archive file in ZOO format |
| .zuu | A .zoo file that's been uuencoded |
| .zu1 | The first file in a split .zuu file |
| .zu9 | The ninth file in a split .zuu file |
| .zu10 | The tenth file in a split .zuu file |
| .uu1 | The first shar file in a (possibly split) .zuu |
| .uu9 | The ninth shar file in a (possibly split) .zuu |
| .uu10 | The tenth shar file in a (possibly split) .zuu |

To get all this back into a usable form, you need to:

1. Unpack the files from the shar format (the verb is usually called 'unshar'). You can either use a standard text editor to remove the info, or the unix 'sh' (Bourne Shell), or a publicly available tool to do it for you. With some tools (like 'sh'), you are required to remove the Usenet header information first; other unshar tools will remove it for you. Once you unshar a .uu? file, you'll have a .zu? file.

2. Combine all the .zu? pieces into a large .zuu file. You should use something like 'cat' under UNIX or 'join' under AmigaDOS. It is important that you combine the pieces in numerical order. Be aware that the single digit pieces do not have a leading zero, so a wildcard sort will probably not do what you want. Instead, you should specify the files in this way: zu? zu??, which will do the single digit files first, then the double digits. You can rest assured there will not be any triple-digit postings.

3. Convert the .zuu file to a .zoo file with the 'uudecode' program. If you have a version of uudecode that understands the checksum and file size information, you should use it. If you don't, you can still use the old uudecode program, but you will not know if the file has somehow become corrupt.

4. You now have a .zoo file. To extract the files into a directory, you need a program called 'zoo'. After "un-zoo'ing", most submissions

will have some documentation to tell how to set up and run the software on your Amiga.

---

Of course you need to get the program(s) to your Amiga to run them. Usually this means downloading the software to your Amiga through a terminal program. If you want to download the files as quickly as possible to your Amiga, you should do steps 1-3 above on your UNIX machine and download the .zoo file to your Amiga, since the .zoo file is the smallest of the files. It's also usually faster and easier to do steps 1-3 on the UNIX machine because you have less worry about disk and memory limitations.

However, some terminal emulators available for the Amiga do not handle binary downloads well. Experiment with yours and see. If you have trouble, you might try a different protocol or different terminal program. If all else fails, you can download the .zuu file, which is an ascii file, and do the uudecode on your Amiga.

The mechanics of terminal programs, protocols and downloading are beyond the scope of this document. Most terminal programs should have a manual section describing how to log in and do remote downloads.

---

Here are the tools you need, under UNIX, AmigaDOS, or both. All are publicly available; you do not need to purchase anything to make use of the software that comes across in comp.binaries.amiga.

```
shar, sh, unshar - a tool to unpack the shell archives, or use an editor
cat or join      - to append the split .zu? files to one .zuu file
uudecode         - to decode the .zuu file to a .zoo file
zoo              - to unpack the ZOO archive.
```

---

From: mace.cc.purdue.edu!ain (Patrick White)
Subject: How to post to comp.[sources|binaries].amiga

I'll give you a quick rundown of the various ways to make submissions in preferred order..

1) make up some postings and post them to the proper groups -- since these groups are moderated, the posting will get sent via mail to the moderator(s). Preferred. Helps us a bit if we forward the submission to a different machine.
2) send mail to the mail aliases amiga-sources-request or amiga-binaries-request. Next best -- but be sure the size of the file will go through mail unscathed (100K or less is usually ok). (BTW, the aliases are always directed to the right person to receive the submissions)
3) send mail directly to the moderators. Same warnings as above, but also be sure to send it to the right person -- if you get it wrong, your mail might never be heard from again.

Since all these ways use mail, the data transmission is equally reliable between them. The main difference is ease of submission, and what happens at our end.

The current moderators are:
Bob Page - page@Sun.COM
- uunet!sun!page

**********

,FNF
From: fnf@fishpond.UUCP
Subject: How to get Fish disks.

WHAT'S AVAILABLE
---

There are "do-nothing-useful" examples of various capabilities of the AMIGA, real development tools, editors, languages, games, and other odds & ends. Also included are machine readable form of many of the examples (received directly from C-A sources) out of the official AMIGA manuals, including the ROM Kernel Manual.

HOW TO OBTAIN DISKS
---

First, check with your local dealers and user groups. Many already have the earlier disks. Since these disks can be copied freely, and widespread distribution is encouraged, they propagate out to central distribution points fairly quickly.

If you just can't wait, or can't find copies locally, I am willing to make these disks available for the cost of media, mailing materials, postage, and miscellaneous expenses (like wear and tear on my drives). My goal is to get as much software as possible into the hands of people that can use and enhance it, and make the AMIGA the success it deserves to be.

Generally, each disk contains all source necessary to recreate the executables provided. All programs are currently compiled with the latest Lattice C and/or Manx C compiler available at the time of release. In a very few cases (noted in the description) the code will not compile or run for some reason, but was considered interesting enough to include anyway. Almost all executables are known to run on the latest kickstart/workbench combination available at time of release.

Disks are typically 85 to 95 percent full.

HOW TO ORDER
---
To order, send a list of the disks you want, and $6 per disk ($5 per disk for 10 or more disks), in U.S. funds, to:

Fred Fish
1835 East Belmont Drive
Tempe, Arizona 85284
USA

Price includes cost of media, mailing materials, and first class domestic postage. Overseas orders add $5 per order for Air Mail. Time and other jobs permitting, all disks will be mailed within 3 days of receipt of order.

Feel free to order more than the current number of disks available. Excess funds will be placed "in escrow" (refundable at any time) and drawn against for automatic mailings of future disks as they become available. I hope to add at least two to four disks per month to the library. Given that I have a database of about 300Mb of freely distributable software to draw upon, that should be a fairly easy goal to accomplish.

DISK CATALOG
---

A disk based catalog is available directly from me at no charge (as long as you supply disks and postage). The catalog has been split into two disks (disk 0A and disk 0B). To get a copy send two disks and $0.65 postage to the above address.

DISTRIBUTION CRITERIA
---

To the best of my knowledge, materials in this library are freely redistributable. This means that they have met one or more of the following conditions:

(1) The materials contains explicit copyright notices permitting redistribution.

(2) The materials were posted to a publically accessible electronic bulletin board and did not contain any copyright notice. (Such materials will be removed if it is subsequently shown that copyright notices were illegally removed.)

(3) The materials were posted to a widely disseminated electronic network (such as usenet), thus implying that their author/poster intended them to be freely distributed. This applies only if they contain no notice limiting distribution.

(4) The materials contain an explicit notice placing them in the public domain. This is not the same as condition (1).

One last note; I get frustrated when I get email from people and then can't get a message back to them because of some sort of braindamage in the return path. Please don't assume that just because you never saw a bounced message, that it actually got through to me, or even if it did, that I ignored it. I DO RESPOND TO ALL MY MAIL. I don't spend ANY time trying to figure out how to work around bad paths if my response gets bounced back. For best results, include your phone number in case the email route fails completely. Also note that my preferred usenet address has changed recently to this machine (fishpond!fnf), rather than mcdsun!fnf.

---

From: ali@polya.stanford.edu
Subject: How to get a listing of Fish disks electronically:

If you can FTP, you can find the following files on SCORE.STANFORD.EDU, in the directory <ALI.AMIGA.FISH>

| | |
|---|---|
| WHEREIS | (a list of programs that've appeared on Fish disks 1..138, sorted by prog name & with one line description) |
| 1, 2, 3, ... | (detailed descriptions of what's on each disk --- these are Fred Fish's own descriptions of what's on the disks) |

You can anonymously FTP these files. Because SCORE is a TOPS-20 machine, you can't CD to the directory, instead you need to:

FTP> get <ali.amiga.fish>whereis

If you want it let me know and I can mail you at least the WHEREIS file.

Ali Ozer, ali@polya.stanford.edu
**********
,PIC
>From: joe@dayton.UUCP
Subject: Re: Picture swap (was Re: clite demo)

So I am hereby officially becoming a picture collector. If you have pictures, drop me a disk. For every disk of pics I get, I'll send it back with what I may have found by that point.

A few notes:

1. If you send ray-traced, sending the input to the ray tracer would be appreciated. That's up to you.

2. If the picture can not be displayed with DPaintII or the DISPLAY program that comes with DBW_Render, please provide a display program somewhere on the disk or at least give me some clues as to how I should display it.

3. Make *sure* you give me full mailing info and such.

4. Anything you send me should be freely redistributable. This means that I'd love to get digitized pictures out of your favorite magazine, but that's not legal.....

5. If you DO violate note 4, at least tell me you did so I don't get into trouble with someone Official.

6. Keep copies and insure yourself if you send more disks than you're willing to lose in the mail.

If I start to get some collections together, I'll do a followup and tell people what they can do to get some pictures from me without sending submissions. I'm also willing to accept pictures over the phones if I don't have to make long long-distance phone calls to do it. If you have DBW_Render pictures, you can just email those to me....

My mailing address is NOT my work address, however. Send disks, etc to:

Joe Larson
6121 St. Croix Ave. N.
Golden Valley, Minnesota 55422

Disks sent to me at work will get here, but home would be better....
--
UUCP: rutgers!dayton!joe  Dayton Hudson Department Store Company
ATT : (612) 375-3537  Joe Larson/MIS 1060
(standard disclaimer...)  700 on the Mall  Mpls, Mn. 55402
**********
,WED
[The following is included to prevent the occasional questions on the net and in NO WAY ADVOCATES OR DISCOURAGES the use of the wedge.]

>From: stan@teroach.UUCP
Subject: Low cost Hard Disk is here for A1000 and A500!!

[stan's opinion on the wedge deleted for objectivity]
So if you don't mind I'll pass on the pertinent information:

R.S.I.
9651 Alexandra Road
Richmond, B.C.
Canada  V6X 1C6
(604) 270-0064

Again, I am in no way affiliated with any of the manufacturers mentioned above, I just want the word to get out, HD RELIEF IS HERE!!
**********
,SID
From: edwin@hcr.UUCP
Subject: Putting the 2000 on its side.

There seems to be no problem with putting the Amiga 2000 on its side to make it fit nicely under a desk. My 2000 has been in this orientation for approximately 10 months and I have had no problem associated with its position. I recommend putting your computer on its left side so that the drives are as high (and accessible) as possible.

You may need some extension cables for the keyboard and mouse. A standard joystick extender cable is fine for the mouse and a standard 7 prong DIN cable will work for the keyboard. Both are available at Radio Shack and similar stores.

There has been some discussion about the operation of disks and drives in a sideways position. I have had no more or no less bad disks because of this. However, it remains to be seen if the disks wear out faster sideways. I have heard of problems with hard disks, however. It seems that some hard disks will only work in the orientation that they were formatted. If you have this problem, you can do a complete backup and reformat your drive, and it should work.
**********
,BUY
From: uunet!Sun.COM!cmcmanis (Chuck McManis)
Subject: a comparison of PC's... which one to buy

This posting is an effort on my part to demonstrate why these arguments are pointless. I try to show that the Amiga 500 and the 1040 ST are essentially the same design with different compromises. When all is said and done they are the same computer. You as a reader look at the two machines, look at the way the designers compromised, and pick the one that *you* like better. Also be aware that your choice is only the better choice from your perspective, others will disagree and they are also correct from *their* perspective.

Take the following comments from Peter as an example :
In article <11767@cup.portal.com> (Peter Ted Szymonik) writes:
>  ...I'd say Atari will have no
> problem reaching the million mark next year if it hasn't already
> done so! ...

Given the "magicalness" of a million machines (see the PS/2, Apple Mac announcements when they broke a million) it is clear that Atari (or Commodore for that matter) will be shouting to everyone that can read (especially developers) to let them in on the good news. Witness Commodore's hyper sensitivity as they get closer to that number.

> Also, I'm sure that a good chunk of those Amiga sales
> were the 500 which was probably bought primarily as a game machine
> while the majority of STs out there are 1040's which have much
> greater utility.

This is an especially wonderful example of why debating machine merits is a waste of time. Here is a guy who obviously is very proud of his computer ownership attempting to slander the "competition" with absolutely no facts at all. If he understood the Amiga computer line he would realize that the Amiga 500 *is* the Atari 1040 ST competition. Line up the features

side by side and even a third grader could tell you they were the same machine, to wit :

| | 1040 ST | Amiga 500 |
|---|---|---|
| Processor | 68000 | 68000 |
| "Main" Memory | 1 Meg | 1 Meg |
| Screen Resolution | 320 X 200 | 350 X 200 |
| various #'s of | 640 X 200 | 700 X 200 |
| colors. | 640 X 480 | 700 X 440 |
| Disk Drive | 720K | 880K |
| Serial Port | Yes | Yes |
| Parallel Port | Yes | Yes |
| MIDI Port | Yes | No |
| DMA Port | Yes | No |
| Expansion Bus | No | Yes |
| Hard Disk Available | Yes | Yes |
| Memory Expansion Avail | Yes | Yes |
| Max Memory | 4 MB | 9 Mb |
| Blitter | No* | Yes |
| Audio | Yes | Yes |

---

* Blitter optional according to some things I have seen

And the astute readers will note that a.) Prices are different, b.) Implementations are different, and c.) Neither machine is the "better" machine to everyone. In terms of hardware capabilities they offer the same functions.

The question you ask are "What are the decisions the designers made?" Take MIDI for example. The Amiga does not have a MIDI port, the designers did not decide to include one. It is easy to turn the Amiga serial port into a MIDI port but it will cost you the user extra cash. *You* decide which decision is better for *you*. Same thing with a hard disk. Atari builds in a DMA port that makes connecting a SCSI device easier, Commodore provides an expansion bus that you can connect a hard disk controller to. Here the Atari designer may have said "Well, either we offer a hard disk interface or an expansion interface, which will it be? Gee most of our users will just want a hard disk so lets offer that." Whereas the Amiga designer might have said "Lets offer an expansion port so that other things beside a hard disk can be easily connected here." The Atari decision makes for cheaper hard disks, the Commodore decision makes for greater flexibility. Which do *you* prefer? *You* decide and that makes that machine best for you. When one evaluates both machines you will notice that the Commodore decisions tended to flexibility even when it raised the cost, whereas Atari's leaned toward keeping the price down at the cost of flexibility. None of these decisions make one computer "better" than the other, they just make the computers different.

Of course none of this means anything to someone who's ego is tied up in the computer they own. That type of person will get just as hyper about whether or not a BMW is better than a Mercedes or a Ford is better than a Mercury. The important point to remember is that when you read an article from someone who really likes his or her computer and thinks anyone who doesn't like their computer obviously has a learning disability, you have to understand that they are *correct*. This is how they think from *their* perspective. There is no need to point out how their perspective disagrees strongly with *your* perspective. That is because both of your perspectives will disagree with everyone else's perspective. The end result is a couple of thousand articles describing the authors perspective and views. If you could condense them into a single survey message you might get the Signal to Noise ratio up above .5 but I doubt it.

Well, thats *my* perspective,

--
--Chuck McManis
uucp: {anywhere}!sun!cmcmanis  BIX: cmcmanis  ARPAnet: cmcmanis@sun.com
These opinions are my own and no one elses, but you knew that didn't you.

---

>From: geoffs@smoke.UUCP
Subject: Re: Atari vs. Amiga: what to buy?

ATARI VS. AMIGA: WHAT TO BUY?

* the bottom line is -- whatever you like best. *
but here are some thoughts:

WORK/SCHOOL OS features:
==========

What kind of environment are you using at work/school? It is Likely Multi-tasking. (VAX VMS, UNIX, ?)

- (AMIGA & ST) If VMS, I don't know of any VMS look-alikes for either machine. there is a PD editor (TVX) that can be configured to look like the VMS EDT editor I think... I know it can be made to look like VI and emacs/jove/uemacs... (unix).

- (AMIGA) If unix at work,... unix editor PD look-a-likes "vi" and "emacs" are available. (They are also available on the ST).

- (ST) If unix at work, then note that an ST version of MINIX has gone to Prentice Hall for distribution (out in Sept? - check comp.os.minix on when). You are familiar with MINIX? It is Version 7 unix look alike muli-tasking OS, comes with compiler and unix-behaving tools.

- (AMIGA) As you know the Amiga has multi-tasking built in, so if you can handle a different interface from the one at work, but are used to multi-tasking, you are already set.

- (ST) UNIX? Today *now* there is a nice PD shell on the ST (gulam) that looks a lot like the unix shell with many of the commands. The editor is emacs-like, and you can pause the editor, escape to the shell, compile a program, then bring the editor back to the fore ground (via "fg" a-la unix). Not multi-tasking. Context switching, but it is the unix look and feel.

- Note: Amiga and ST can run MSDOS stuff with appropriate add-on's. ST cam also run MAC stuff with appropriate add-on's.

MIDI:
=====

The ST came with midi's built in and this has generated a lot of ST midi software. Buy a copy of STart magazine. There are always ads in that artist-oriented magazine for music editors, and multiple other midi programs. I do not know what midi stuff is available on the AMIGA. I think there is a large library of midi software on the PC market. You may wish to look here too...

Final suggestions:

1. Get several machine specific magazines on both computers. Try to stay away from those published by the machine manufacture; they tend to be censored and over-hyped: "See how great we are!!!!!!!!".

2. Please try before you buy. - you may hate the feel of the keyboard.

*BOTH* good computers, you probably can't go wrong what ever you choose. To be fair, I do not own an Amiga. I own 2 Commodors (the VIC-20 & C64) am the happy owner of 1 Atari (the ST).

I will not suggest either computer over the other. That is a personal preference. A choice only you can make. My experience with both companys is about the same. They are probably on a par with the rest of the retail computer industry.

--
---> geoffs@brl.arpa
--

---

>From: AmigaIan@cup.portal.com.UUCP
Subject: Re: 500 or 2000 ?

>My next question is this:
>should I buy an amiga 500 or an amiga 2000?
>Can someone give me the pro's and con's of the two machines?

Ok, let me give it a try, The 2000 comes with 512K more than the 500 but I have seen 512K expansion for the 500 priced under $150.
You can have IBM compatibility with the 2000 but not with the 500.
The 2000 costs more $$$ but it is a little less expensive to expand.
The 2000 is bigger than the 500.
If you expand with the 500 you need alot of space.

The 500 is about $700 cheaper if you get a 1 meg 500.
The 2000 is now selling for $1499.
The 500 is now selling for $649.

Did I miss anything? I believe those are all the diffrences. Hope I made things easier rather than harder.

Ian_Matthew_Smith@cup.portal.com
**********
,CPU
>From: daveh@cbmvax.UUCP
Subject: Re: Is 14.32MHz too fast for my expansion RAM?

> I don't know about the CMI board, but some 68000 speedup boards have
> floating point processor sockets. If you use it, and are running
> programs with a lot of FP computation that support it, you should
> get a dramatic speed increase, yes/no?

You'll get a speedup over no 68881 at all (assuming 1.3 math libraries). What you don't get is a 32 bit interface to the 68881, or a true coprocessor interface to the 68881, so this is still considerably slower than a similarly clocked 68020. Given the speedups you can get with any kind of math chip, it's probably a good idea for someone into image rendering or other heavy duty math operations who doesn't have a 68020 type budget.

> If you do a lot of floating point stuff (and a lot of graphics programs do),
> it sounds like a lot more bang for the buck compared to a 68020 board
> with coprocessor at ~$1000 more (plus 32 bit memory to get a real advantage
> over the 68000).

Note that for math operations, a 68020 with 68881 is several times faster than a 68000 with 68881, even on a 16-bit bus. The difference is of course the CPU interface that the 68020 uses, plus the wide data bus. The 68000 or 68020 have to fetch data for the 68881 in either case. But while the 68000 will fetch at 16 bits, then transfer at 16 bits, the 68020 will do its transfers at 32 bits. Of course, if you can only afford a fast 68000 board at $200-$300, it's going to speed up your math more than that cash sitting in the bank waiting for a 68020 board will...

Dave Haynie "The 32 Bit Guy" Commodore-Amiga "The Crew That Never Rests"
{ihnp4|uunet|rutgers}!cbmvax!daveh PLINK: D-DAVE H BIX: hazy
"I can't relax, 'cause I'm a Boinger!"
**********
,LED
>From: bjc@pollux.UUCP
Subject: Re: What does flashing caps lock LED mean?

In article <1200@tellab5.tellabs.CHI.IL.US> grj@tellab5.UUCP () writes:
>Sorry if this has been covered many times before, but I don't get a chance
>to read this group often. Anyway - sometimes when I power on my A2000,
>the caps lock LED flashes at about a 1 or 2 Hz rate. The keyboard is then
>useless and nothing typed gets recognized. It seems to have started

Look on page 3 of Appendix H to the Hardware Reference manual.
Briefly, though, it says:

| | |
|---|---|
| One blink | Keyboard ROM test failed |
| Two blinks | Keyboard RAM test failed |
| Three blinks | Watchdog timer test failed |
| Four blinks | A short exists between two row lines or one of the seven special control keys. |

It also says that you probably would not have been able to type anything before this sequence would be completed, but if so, the key codes would be sent to the computer and then a "terminate key stream" code would be sent. There is more information in that appendix.

Betty
**********
,MAC
>From: daveh@cbmvax.UUCP
Subject: Re: A2000 vs B2000

> How can I tell which one my 2000 is? A or B. Mine has Amiga 2000 in
> little letters on the system box as opposed to the new ones I've seen with
> Amiga 2000 in BIG letters.

If you bought it in the USA, it's a "B" (or someone hosed you). The easiest way to tell is to look at the phono jacks on the back of the machine. The "A" has two, the "B" has three (that extra one is a monochrome video output that I got basically for free when using the A500 video hybrid).

Most stuff works in both. What doesn't probably can't be easily added, though with a minor modification to the case, a board swap is possible.

> I got it in late '87 (December I think maybe Jan '88) and the writing on
> the box is mostly in German.

We kept using boxes from Germany long after switching to the US designed motherboard.

> Will all the hardcards/SCSI interface/genlock/whatever cards fit in the
> A2000?? Or am I screwed there too?

As long as 100 pin cards are in spec, they'll work fine on both machines. All Genlocks should work similarly. The B2000 video slot has many more signals, but the only thing I know of on the market that's currently taking advantage of that slot is MicroWay's flickerFixer, which won't work on the "A" board.

> If I've got the 68020 in my 2000 (which I don't yet) what would be the
> purpose of booting from the 68000???

I put that feature in to allow developers who develop on the 68020 based Amiga an easy way to test their software on a 68000. It has the side effect of allowing you to run software on your machine that doesn't work with the 68000. I've found a better use for such software -- reformatting the disks it comes on. Really, though, there's very little software that can't be used on the 68020.
--
Dave Haynie "The 32 Bit Guy" Commodore-Amiga "The Crew That Never Rests"
{uunet|pyramid|rutgers}!cbmvax!daveh PLINK: D-DAVE H BIX: hazy
Amiga -- It's not just a job, it's an obsession
**********
,SUR
From: Jim Carson <uunet!rice.edu!carson>
Subject: comp.sys.amiga survey

I BACKGROUND
a. How experienced do you consider yourself with the Amiga?

EXPERIENCE:
Very -- wrote popular software 6 had amiga > 2 years 10
Fair amount of programming. 4 had amiga > 1 year 4
Some Programming 5 had amiga < 1 year 2
No programming on Amiga 10 did not say 10

b. What kind of hardware do you have? (Some people had more than one system)

Machines:
A1000 17 A/B2000 9 A500 7

Peripherals:
External Memory 18 Extra Floppy 14
Modem 11 Hard Disk 10
Printer 10 Accelerator board 3
Sound/MIDI interface 3 Clock Card 2
Sidecar 2 Digitizer 1
Ethernet Interface 1

c. How much do you use your Amiga? (Some people said "lots", which I took to mean 2 - 4 hours/day)
> 4 hours a day 10
2 - 4 hours/day 6
1 - 2 hours/day 4
< 1 hour/day 1

d. What are your favorite kinds of programs (games, utilities, etc)?
Games 13
Utilities 9
Compilers/Program Lang 8
Telecommunication 6
Music 5
Demos 4
Editors 3

II. HARDWARE
a. Hard Drive/Card
Controllers:
WEDGE 1000; WEDGE 500 ($175-$200) - very good
Bear Products - good ($120 w/o chips)
Commodore 2090; C.Ltd.; Overdrive;- okay
Microbotics HardFrame
Startime - Bad performance

Drives:
Miniscribe; Seagate ST157N, ST277N - okay
Quantum; Rodime; Supra SCSI; GVP Impact; CDC Wren - good

b. Other Permanent Storage (Tape, CD, etc)

SCSI tape is nice for backup. R/W opticals are great for animation (650MB Sony).

c. Modem
Supra 2400 - Great ($150 for 2400, $70 for 1200)
Hayes - Nice, but expensive.
Trailblazer T2000+; Ventel MD212 - good
Courier 2400; BytCOM 212AD; - okay
Bizcomm Intellimodem EXT; Avatex
1680 - adequate (incompatible w/software)

d. Midi interfaces
ECE MIDI interface, connected to an Ensoniq Mirage keyboard.
Golden Hawk MIDI Gold MIDI interface. - Nice little box, does the job

f. External Disk Drives

A1010. (3.5") - okay
A1020 (5.25") - slow, but good for transferring ibm <-> amiga
California Access - good
Byte by Byte PAL Jr. - nice, but obsolete. Company sucks.

g. Printer

Great:
Epson LQ-800. Nice, but not as good as an Imagewriter on a Mac
C.Itoh C310-xp - Fast (300cps), Very nice NLQ
Panasonic KX-P1090i - Very nice NLQ ($200)

Ok:
Epson LX-800 - cheap, quick with 1.3 drivers, graphics
Howtek PixelMaster color inkjet
IBM Colorjet
Okidata 82A w/Rainbow Tech firmware upgrade
Mannesman-Tally Spirit-80 - Excellent printer 5 years ago, good company.

Marginal quality:
Okimate 20 - cheap printer
Star SG-10 - cheap printer
Epson FX-286 - slow. paper jams incessantly. ugly NLQ.
Citizen msp10 - slow. problems with proportional print & tabs.

h. Monitor

Sony KV1311 - very good
C-A 1080 - good
C-A 2002 - good
NEC Multisync - good
Thompson 4120 - good
C-A 1084 - okay, but crackles
C1702 [modified] - adequate

i. Memory Expansion

INSIDER board (1MB + RT Clock) - great
Spirit, A501 - good
Starboard 2-meg - good
Orphan 4-meg - no recommendations
**********

,CLI
From: hcr!edwin
Subject: Closing the initial CLI

% "I can't get my initial CLI window to close" seems to be a commonly
% asked question. It would be good to explain "run" vs. "runback"

Okay, a CLI will not close if a program has been "run" from it. (Check your startup-sequence) This is because the program that has been "run" needs someplace to get and put its I/O. If your program does not need to do I/O to the CLI, you should use the RunBack program from Fish 152 (earlier version on Fish 65), or RunBackGround on Fish 73. These programs will totally detach your program from the CLI and allow you to close it normally with endcli.

Edwin
**********
,KIK
From: bryce@cbmvax
Subject: Re: Crashing all the way to Kickstart (*the solution*)

In several articles <> several.net.posters wrote:
> [Since installing V1.3 I have noticed something strange. If the
> machine ever crashes, it sometimes goes back to the Kickstart hand
> instead of the Workbench hand.]

Thank you all for noting and reporting this bug. I have located the cause; a code mistake causes the Kickstart protected memory to be left open for writing during some types of system Alert. This problem will be addressed in the next version of "SetPatch" (no date or distribution details are available at this time).
**********
,HAR
From: Witting@topaz.rutgers.edu
Subject: Hard drives for Amiga (especially the 2000)

NOTE: The following is a summary of replies concerning my future purchase of an Amiga 2000. I want a Hard Disk, and was inquiring into advantages/disadvantages of the different cards, esp in ref to the new Amiga 2000HD, which came with a CBM 2090A cantroller and HD The 2000HD will sell about $100 less than a system built with third party components. Of course, shop around to find the price diff you will actually be working with.

BTW, tech note. SCSI interface transmits 1 byte at a time, ST506 1 bit at a time. ST506 is the IBM standard, SCSI looks to be the new standard.

2000HD- Basic a 2000 with a CBM 2090A Hard Disk Controller and a 40meg HD connected through a ST506 interface, leaving one more ST506 and 6 SCSI connectors open. The 2090A uses DMA to control the hard disk, leaving the processor free to do other things.

The GVP HardCard is heavily recommended, clearly the best of the established third Party cards. It has been around a while, so all the bugs are gone, but it supposedly relies upon the processor to do a lot of the work. The Quantum Drives are supposedly the Best on the market, and is available in 28ms and 11ms types. Expect it to last twice as long as typical drives. May require EEPROMS to Autoboot.

The Microbotics card is impressive, claiming to transfer data at bus speeds (as fast as the Amiga can handle it, not 65mph hiway, 15 mph city ;) It uses DMA to control the drives, leaving the processor free like the 2090A. It is a half sized card, so it can be used in expansion boxes, has power hookups for a card mounted drive, and several extra SCSI interfaces.

There are a few other points to consider:

1. The CBM card has its own processor, the GVP can bog the processor down if you are working processor intensive tasks, otherwise the GVP is faster. The microbotic is like the CBM card in that it doesnt harass the processor, and is the fastest and newest design of the three.

2. The CBM comes pre-installed. The cards should not be hard to install, but I understand damage done by improperly installed cards is not covered by warranty. But the 3rd party warrantees are typically longer, CBM offers only a 90 dayer. This does not apply if

you already own an Amiga, but having an authorized service rep install it should keep warranty problems away. It can also put you out another 60 or 70 bucks.

3. The CBM must boot to the old file system, not the newer fast filing system. While this only affects a small section of the Disk (The rest can be configured fast filing) The others don't have this weakness. K1.3 was designed for autobooting to HD's but some GVP cards may still require additional PROMS.

Personally, I'd like the microbotics with the Quantum 11ms, but 1st I gotta check prices. I also want to investigate the possible bug. *I have heard from a few people who own the microbotics, and so far they are extremely happy with them*

paul

--
"Run, Run, as fast as you can, you cant catch me, Im the gingerbread man"
"Lets do some crimes. Yeah, lets get sushi and not pay"
6 6
Witting@topaz.rutgers.edu `--'

**********

,COL
>From: hah@mipon3.intel.com (Hans Hansen)
Subject: All dem colors during boot

A M I G A ' S  I N I T I A L I Z A T I O N  C O L O R S

| | |
|---|---|
| Dark gray | Ok Hardware |
| Light gray | Ok Software |
| Red | Bad ROM Checksum |
| Green | Bad RAM |
| Blue | Bad Custom Chips |
| Yellow | Exception |

A M I G A ' S  I N I T I A L I Z A T I O N

Clear Chips
Disable DMA and Interrupts
Clear the Screen
Check the Hardware
Pass or fail the Hardware to the Screen
Checksum the ROMs
Pass or fail the ROMs to the Screen
System setup
Check for RAM at $C00000
Move SYS_BASE to $C00000 if it exists
RAM Test
Pass or fail the RAM to the Screen
Check the Software
Pass or fail the Software to the Screen
Set up the RAM
Link the Libraries
Find External RAM and link it to the list
Set up Interrupts and DMA
Start default Task
Check for 68010, 68020, and 68881
Check for an Exception
System Reset

Hans

**********

,GUR
>From: bryce@hoser.berkeley.edu (Bryce Nesbitt)
Subject: Re: GURU-NUMBERS

[this was posted before Bryce was hired by C-A and I kept it - ed]

This comes from the V1.2 include files. Save this file if you think you might ever want it.

```
/*********************************************************************
*  Format of the alert error number:
*    +-+-------------+---------------+-------------------------------+
*    |D|  SubSysId   |  General Error |    SubSystem Specific Error   |
*    +-+-------------+---------------+-------------------------------+
*         D:  DeadEnd alert
*        SubSysId:  indicates ROM subsystem number.
*   General Error:  roughly indicates what the error was
*  Specific Error:  indicates more detail
*********************************************************************/

/*  General Dead-End Alerts:*/

/*------ alert types */
#define AT_DeadEnd  0x80000000
#define AT_Recovery  0x00000000

/*------ general purpose alert codes */
#define AG_NoMemory  0x00010000
#define AG_MakeLib  0x00020000
#define AG_OpenLib  0x00030000
#define AG_OpenDev  0x00040000
#define AG_OpenRes  0x00050000
#define AG_IOError  0x00060000
#define AG_NoSignal  0x00070000

/*------ alert objects: */
#define AO_ExecLib  0x00008001
#define AO_GraphicsLib  0x00008002
#define AO_LayersLib  0x00008003
#define AO_Intuition  0x00008004
#define AO_MathLib  0x00008005
#define AO_CListLib  0x00008006
#define AO_DOSLib  0x00008007
#define AO_RAMLib  0x00008008
#define AO_IconLib  0x00008009
#define AO_ExpansionLib 0x0000800A
#define AO_AudioDev  0x00008010
#define AO_ConsoleDev  0x00008011
#define AO_GamePortDev  0x00008012
#define AO_KeyboardDev  0x00008013
#define AO_TrackDiskDev 0x00008014
#define AO_TimerDev  0x00008015
#define AO_CIARsrc  0x00008020
#define AO_DiskRsrc  0x00008021
#define AO_MiscRsrc  0x00008022
#define AO_BootStrap  0x00008030
#define AO_Workbench  0x00008031

/* Combine the above to get the actual number */

/*  Specific Dead-End Alerts: */

/*------ exec.library */
#define AN_ExecLib  0x01000000
#define AN_ExcptVect  0x81000001  /* 68000 exception vector checksum */
#define AN_BaseChkSum  0x81000002  /* execbase checksum */
#define AN_LibChkSum  0x81000003  /* library checksum failure */
#define AN_LibMem  0x81000004  /* no memory to make library */
#define AN_MemCorrupt  0x81000005  /* corrupted memory list */
#define AN_IntrMem  0x81000006  /* no memory for interrupt servers */
#define AN_InitAPtr  0x81000007  /* InitStruct() of an APTR source */
#define AN_SemCorrupt  0x81000008  /* a semaphore is in illegal state */
#define AN_FreeTwice  0x81000009  /* freeing memory already freed */
#define AN_BogusExcpt  0x8100000A  /* illegal 68k exception taken */

/*------ graphics.library */
#define AN_GraphicsLib  0x02000000
#define AN_GfxNoMem  0x82010000  /* graphics out of memory */
#define AN_LongFrame  0x82010006  /* long frame, no memory */
#define AN_ShortFrame  0x82010007  /* short frame, no memory */
#define AN_TextTmpRas  0x02010009  /* text, no memory for TmpRas */
#define AN_BltBitMap  0x8201000A  /* BltBitMap, no memory */
#define AN_RegionMemory 0x8201000B  /* regions, memory not available */
#define AN_MakeVPort  0x82010030  /* MakeVPort, no memory */
#define AN_GfxNoLCM  0x82011234  /* emergency memory not available */

/*------ layers.library */
#define AN_LayersLib  0x03000000
#define AN_LayersNoMem  0x83010000  /* layers out of memory */

/*------ intuition.library */
#define AN_Intuition  0x04000000
#define AN_GadgetType  0x84000001  /* unknown gadet type */
#define AN_BadGadget  0x04000001  /* Recovery form of AN_GadgetType */
#define AN_CreatePort  0x84010002  /* create port, no memory */
#define AN_ItemAlloc  0x04010003  /* item plane alloc, no memory */
#define AN_SubAlloc  0x04010004  /* sub alloc, no memory */
```

```
#define AN_PlaneAlloc  0x84010005  /* plane alloc, no memory */
#define AN_ItemBoxTop  0x84000006  /* item box top < RelZero */
#define AN_OpenScreen  0x84010007  /* open screen, no memory */
#define AN_OpenScrnRast 0x84010008  /* open screen, raster alloc, no memory */
#define AN_SysScrnType  0x84000009  /* open sys screen, unknown type */
#define AN_AddSWGadget  0x8401000A  /* add SW gadgets, no memory */
#define AN_OpenWindow  0x8401000B  /* open window, no memory */
#define AN_BadState  0x8400000C  /* Bad State Return entering Intuition */
#define AN_BadMessage  0x8400000D  /* Bad Message received by IDCMP */
#define AN_WeirdEcho  0x8400000E  /* Weird echo causing incomprehension */
#define AN_NoConsole  0x8400000F  /* couldn't open the Console Device */


/*------ math.library */
#define AN_MathLib  0x05000000

/*------ clist.library */
#define AN_CListLib  0x06000000

/*------ dos.library */
#define AN_DOSLib  0x07000000
#define AN_StartMem  0x07010001  /* no memory at startup */
#define AN_EndTask  0x07000002  /* EndTask didn't */
#define AN_QPktFail  0x07000003  /* Qpkt failure */
#define AN_AsyncPkt  0x07000004  /* Unexpected packet received */
#define AN_FreeVec  0x07000005  /* Freevec failed */
#define AN_DiskBlkSeq  0x07000006  /* Disk block sequence error */
#define AN_BitMap  0x07000007  /* Bitmap corrupt */
#define AN_KeyFree  0x07000008  /* Key already free */
#define AN_BadChkSum  0x07000009  /* Invalid checksum */
#define AN_DiskError  0x0700000A  /* Disk Error */
#define AN_KeyRange  0x0700000B  /* Key out of range */
#define AN_BadOverlay  0x0700000C  /* Bad overlay */

/*------ ramlib.library */
#define AN_RAMLib  0x08000000
#define AN_BadSegList  0x08000001  /* no overlays in library seglists */

/*------ icon.library */
#define AN_IconLib  0x09000000

/*------ expansion.library */
#define AN_ExpansionLib 0x0A000000
#define AN_BadExpansionFree  0x0A000001

/*------ audio.device */
#define AN_AudioDev  0x10000000

/*------ console.device */
#define AN_ConsoleDev  0x11000000

/*------ gameport.device */
#define AN_GamePortDev  0x12000000

/*------ keyboard.device */
#define AN_KeyboardDev  0x13000000

/*------ trackdisk.device */
#define AN_TrackDiskDev 0x14000000
#define AN_TDCalibSeek  0x14000001  /* calibrate: seek error */
#define AN_TDDelay  0x14000002  /* delay: error on timer wait */

/*------ timer.device */
#define AN_TimerDev  0x15000000
#define AN_TMBadReq  0x15000001  /* bad request */
#define AN_TMBadSupply  0x15000002  /* power supply does not supply ticks */

/*------ cia.resource */
#define AN_CIARsrc  0x20000000

/*------ disk.resource */
#define AN_DiskRsrc  0x21000000
#define AN_DRHasDisk  0x21000001  /* get unit: already has disk */
#define AN_DRIntNoAct  0x21000002  /* interrupt: no active unit */

/*------ misc.resource */
#define AN_MiscRsrc  0x22000000

/*------ bootstrap */
#define AN_BootStrap  0x30000000
#define AN_BootError  0x30000001  /* boot code returned an error */

/*------ Workbench */
#define AN_Workbench  0x31000000

/*------ DiskCopy */
#define AN_DiskCopy  0x32000000
```

**********

,HED
From: jesup@cbmvax
Subject: Re: 15 Head Hard drive and the A2090(a)

In article <1735@dciem.dciem.dnd.ca> kevin@dretor (Wallace B. Wallace) writes:
>I recently acquired a 15 head Hard Drive for my Amiga 2000 only to discover
>that the A2090(a) does not support >8 heads. Has anyone figured out either
...
> BTW, the drive is a Fujitsu 2249SA, SCSI, 320M, 8 disks, 15 heads

Maybe this should be added to the introduction to comp.sys.amiga.

The A2090 supports 8 heads for ST506 drives. For SCSI, the number of heads is not important, so it WILL support all of your drive.

--
Randell Jesup, Commodore Engineering {uunet|rutgers|allegra}!cbmvax!jesup

**********

,ARC
From: raz%kilowatt.EBay@Sun.COM (Steve -Raz- Berry)
Subject: How to use the kilowatt archive server

The following message is the reply that kilowatt (the archive server) will mail to you if you ask it for help. Please be sure that you read it carefully and understand it completely before you post to the net asking for help. If you have problems with the server, and you *can* reach the machine (but it isn't doing what you want) send the symptoms of the problem to manage%kilowatt@sun.com and I'll get back to you as soon as I can.

Also, if anyone is interested in setting up their very own archive server, contact me at the above address and I'll send you all information and programs that you need to set one up.

-Raz

PS: For those that are interested, kilowatt is a Sun 3/260 8meg ram, with 128meg of disk allocated (currently) for the archive.

---

This message comes to you from the archive server at kilowatt@sun.com, archive-server%kilowatt@sun.com. It received a message from you asking for help.

The archive server is a mail-response program. That means that you mail it a request, and it mails back the response.

The archive server is a very dumb program. It does not have much error checking. If you don't send it the commands that it understands, it will just answer "I don't understand you".

The archive server has 4 commands. Each command must be the first word on a line. The archive server reads your entire message before it does anything, so you can have several different commands in a single message. The archive server treats the "Subject:" header line just like any other line of the message. You can use any combination of upper and lower case letters in the commands.

The archives are organized into a series of top-level directories. The top-level index gives you the index for each of the files in the archive.

---

If you are bored with reading documentation and just want to try something, then send the server a message containing the line

send index

When you get the index back, it will give you the names of all of the files in the archive; send the server another message asking it to send you the files that you want:

```
send applications plplot.2 plplot.3
```

etc. If you are using a mailer that understands "@" notation, send to archive-server%kilowatt@sun.com. If your mailer deals in "!" notation, try sending to {someplace}!kilowatt.sun.com!archive-server, e.g. uunet!kilowatt.sun.com!archive-server. For other mailers, you're on your own.

For BITNET mailers and those lazy typists out there, I have aliased archive-server to archive and archive-management to manage. This means that you can also use the net address of:

| | |
|---|---|
| archive%kilowatt@sun.com | {}kilowatt.sun.com!archive |
| manage%kilowatt@sun.com | {}kilowatt.sun.com!manage |

---

The server has 4 commands:

"help" command: The command "help" or "send help" causes the server to send you the help file. You already know this, of course, because you are reading the help file. No other commands are honored in a message that asks for help (the server figures that you had better read the help message before you do anything else).

"index" command: if your message contains a line whose first word is "index", then the server will send you the index of the contents of the archive. For example, you can say

```
index
```

You can then send back another message to the archive server, using a "send" command (see below) to ask it to send you the files whose name you learned from that list.

The index is updated nightly, to reflect the contents of the archive. So if you are getting messages that inform you that the file you requested does not exist, ask for the index again and double check the file name.

If your message has an "index" or a "send index" command, then all other "send" commands will be ignored. This means that you cannot get an index and data in the same request. This is so that index requests can be given high priority.)

"send" command: if your message contains a line whose first word is "send", then the archive server will send you the item(s) named on the rest of the line. To name an item, you give its directory and its name. For example

```
send workbench ptranim.uu2
```

or

```
send audio vclock.uu
```

Once you have named a category, you can put as many names as you like on the rest of the line; they will all be taken from that category. For example:

```
send exec xoper13.1 xoper13.2 xoper13.uu1
```

Each "send" command can reference only one directory. If you would like to get files from more than one directory, you must use two "send" commands.

One of the most common mistakes that a new user makes when formatting a send command is to type something that looks like:

```
send dos/fs pthass.sh1 dos/fs pthass.sh2
(This is incorrect... )
```

This will not work for 2 reasons. One, simply because the archive will look for a directory called "dos/fs". All that you need to do is:

```
send fs pthass.sh1 pthass.sh2
(This is correct)
```

The second reason that the previous send command would not work is that you may only specify one path per send command.

You may put as many "send" commands as you like into one message to the server, but the more you ask for, the longer it will take to receive. See "FAIRNESS", below, for an explanation. Actually, it's not strictly true that you can put as many "send" commands as you want into one message. If the server must use uucp mail to send your files, then it cannot send more than 100K bytes in one message. If you ask for more than it can send, then it will send as much as it can and ignore the rest. Since many files in the archive are around 60K, it's probably best to ask for one file at a time unless you know it's safe to do otherwise.

"path" command: The "path" command exists to help in case you do not get responses from the server when you mail to it.

Sometimes the server is unable to return mail over the incoming path. There are dozens of reasons why this might happen, and if you are a true wizard, you already know what those reasons are. If you are an apprentice wizard, you might not know all the reasons but you might know a way to circumvent them.

If you put in a "path" command, then everything that the server mails to you will be mailed to that address, rather than to the return address on your mail. For example, if you say

```
path decwrl!pyramid!rutgers!zakkaroo!jj@uunet.uu.net
```

then all mail sent by the server will be sent to that address. If you use mixed-mode addresses (! and @), the archive-server will put precedence on '@' before '!'.

You cannot expect the archive server to pick a uucp path to be determined for you. If you can't determine a path yourself, make the path relative to a 'known' site, e.g.:

```
path place!person@uunet.uu.net
```

which will cause the archive-server to send to site uunet.uu.net with the instructions "send this to place!person".

EXAMPLES:

1) Find out the list of all of the Amiga files that are in the archive. Send this message:

```
To: archive-server%kilowatt@sun.com
Subject: hi there

index
```

2) Get files from the archive (you have learned their file names from the list that was sent to you in step 1).

```
To: archive-server%kilowatt@sun.com
Subject: send digest 3.17

send iff gif2iff.uu2 ifflib161.uu1
send iff dplaz.uu1
```

(it turns out these three files add up to less that 100k, so they can all be sent by return mail).

3) Get a file, and send it over the best path to my site:

```
To: uunet.uu.net!mcvax!kilowatt.sun.com!archive-server

path myname@site.uucp
send iff iff2ps20.1
```

NOTES:

The archive server acknowledges every request by return mail. If you don't get a message back in a few days (depending on how close you are to sun.com on the network) you should assume that something is

going wrong, and perhaps try sending another request, this time with a "path" command. If you aren't getting anywhere and you don't know a wizard to help you, try putting

```
path mysite!myname@uunet.uu.net
```

in your message, where "myname" is your mailbox name and "mysite" is the uucp name of your machine.

The delays in sending out large items from the archives are intentional, to make it difficult to get copies of everything in the archives. If you are new to the network and would like to get all back issues of everything, you should post a request to a regional newsgroup asking whether someone who is geographically near you can provide them.

Don't send mail with long lines. If you want to ask for 40 files in one request, you don't need to put all 40 of them in one "send" command. The archive server is quite able to handle long lines, but before your mail message is received by the archive server it might pass through relay computers that will choke on long lines, or chop them up.

The archive server does not respond to requests from users named "root", "system", "daemon", or "mailer". This is to prevent mail loops, if your name is "Bruce Root" or "Joe Daemon". Yes, I know about Norman Mailer and Waverley Root. Norman doesn't use netmail and Waverley is dead.

FAIRNESS:

The archive server contains many safeguards to ensure that it is not monopolized by people asking for large amounts of data. The mailer is set up so that it will send no more than a fixed amount of data each day. If the work queue contains more requests than the day's quota, then the unsent files will not be processed until the next day. Whenever the mailer is run to send its day's quota, it sends the requests out shortest-first.

If you have a request waiting in the work queue and you send in another request, the new request is added to the old one (thereby increasing its size) rather than being filed anew. This prevents you from being able to send in a large number of small requests as a way of beating the system. If you request 10 files together, you will get substantially higher priority than if you make 10 requests for 1 file each.

The reason for all of these quotas and limitations is that the delivery resources are finite, and there may be many people who would like to make use of the archive.

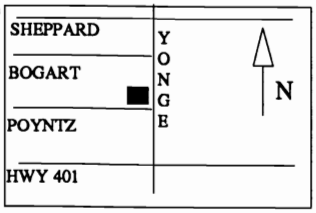
[end of help]

**********


CANADA'S BEST SELECTION OF
AMIGA - C64 - C128 SOFTWARE
COMSPEC HARD DRIVES AND BERNOULLI DRIVES
AVAILABLE FOR
THE AMIGA 500 AND 2000

Electronics 2001
COMPUTER PRODUCT CENTRE

4736 YONGE STREET
WILLOWDALE ,ONT
M2N 5M6
TEL: (416) 229-2700




AUTHORIZED COMMODORE DEALER SINCE 1979

## Product Review

**Media Line Amiga Clipart Disk 1**
Free Spirit Software

suggested retail price $39.95 US

**by Derek Grime**

This collection of black and white, brush size picture files are intended to be used in desktop publishing. Designed by Loren Lovhaug (Mr. Lovhaug is the publisher of the C128-specific magazine *Twin Cities-128*), the value of Media Line's Clipart is questionable.

Media Line, for better or worse, has chosen quantity over quality. There are over two hundred files on the one disk set. This means that each image costs only pennies. You get what you pay for.

The caliber of graphics varies wildly. They range from excellent to very poor. Some, like a picture called "breakfast tray", are detailed line illustrations. The tray contains a full course meal with a mug of coffee and a bouquet of flowers. Others like a text graphic called "It's A Great Day" are so messy they are unusable.

Most of the images appeared to have been scanned in with NewTek's Digi-view using its line art mode. This would be fine, but Media Line has done nothing to clean up the graphics for us. Consequently, the Clipart contains lots of stray pixels and video noise. Borders and rectangles are sometimes skewed creating bad stair-stepping problems. This renders them virtually useless to any professional application.

Few of the images seem to have been drawn directly with Amiga paint programs. Almost half of the Clipart appears to have been scanned in from those Victorian public domain art books that are widely available. Stylistically, these pictures do not transfer well to computer graphics.

The subjects represented cover all areas. There are directories of pictures for sports, celebrities, Christmas and many others. Unfortunately, the collection seems to have an obsession with military vehicles. They are everywhere.

When was the last time you needed a picture of a "Battle Snorkel" or an attack machine called "Big Fred"? There are dozens of these files. Most of them are pointless unless you are doing some sort of undercover cover DTP work for the US Army. Stranger things happen.

If your desktop graphics demands are not too serious, Media Line's Clipart could be for you. The sheer volume of pictures gives the collection some worth. Used prudently, they could add a little life to novice publications. If you require anything more than the basics, keep looking.

# Allocate And Deallocate

*These underused functions provide one answer to fragmentation*

**by Nick Sullivan**

*Nick Sullivan was a founding editor of **Transactor for the Amiga**, and is still involved with the magazine as an associate editor. With long-time partner Chris Zamara, he now runs AHA!, an Amiga software and documentation company. Among other products, his collaboration with Zamara has produced Transcript, Gold Disk's word processor. Nick may be reached on CompuServe (76703,4353) or by telephone at AHA!'s Victoria, BC headquarters (604-721-4102).*

There are so many approaches to memory allocation on the Amiga that the sheer range of choices can be overwhelming. Some programmers tend to stick with one or two familiar allocation calls and leave the others alone, a tactic that makes life simpler in some ways but isn't likely to produce optimum code.

An often-neglected option for memory allocation is the pair of Exec library functions *Allocate* and *Deallocate*. These functions operate at a lower level than the more familiar *AllocMem/FreeMem* (Exec), *AllocRemember/FreeRemember* (Intuition) and *malloc/free* (C library) pairs, and in fact are ultimately called by Exec whenever one of the higher-level functions is used. Even so, there are situations in which it makes sense to use *Allocate* and *Deallocate* directly.

### Memory fragmentation

Fragmentation is the operating system equivalent of aging: in the long run, there's no way to avoid it, but there are things you can do either to accelerate or to retard the process. Retarding it is a worthwhile goal. As Eric Giguère demonstrated in this magazine not long ago, it may be impossible to allocate even a modest amount of space in a badly fragmented system despite an ample total amount of free memory.

Fragmentation on the Amiga is a consequence of multitasking. In a typical scenario, programs A and B are both allocating many chunks of memory for their own use. Because the programs are running 'simultaneously', their requests for memory are interleaved, and the blocks of memory they receive are interleaved also. Now program A exits, duly deallocating the memory it has been using, but program B is still running. Until program B also exits, the total of program A's memory – though now free to be reallocated when program C arrives on the scene – is distributed in many small chunks, and is no longer available as a larger block.

There are many variations on this theme, and many possible complications, but the moral is simple: when possible, it is preferable to allocate the memory you need as a single large block than as many smaller blocks.

### The free memory list

Exec, the operating system component responsible for memory allocation (and much else), does not keep track of the memory it has doled out to its client tasks. It *does* keep track of the free memory remaining, however, by means of a *List* that originates in the *ExecBase* Library structure.

This free memory List chains together a set of *MemHeader* structures, each of which governs a contiguous address space within the computer. For instance, all Amigas have a range of 'chip' memory (available to the custom chips) that typically occupies the lowest half a megabyte of address space in current machines; one *MemHeader* is assigned to this range. Many Amigas also have 'fast' memory in various configurations; each set of contiguous addresses in this kind of memory will also have its own *MemHeader* in the free memory List.

The *MemHeader* structure is defined for C programmers in the include file *exec/memory.h*. It looks like this:

```
struct MemHeader {
   struct Node mh_Node;
   UWORD  mh_Attributes;      /* characteristics    */
   struct MemChunk *mh_First; /* first free region  */
   APTR   mh_Lower;           /* lower memory bound */
   APTR   mh_Upper;           /* upper bound+1      */
   ULONG  mh_Free;            /* total free bytes   */
};
```

Most of the components of this structure are readily understood: the *mh_Node* field provides the link into the free memory List; *mh_Attributes* records the type of memory (MEMF_CHIP, MEMF_FAST and so on) to be matched against the

requirements specified in an allocation request; *mh_Lower* and *mh_Upper* give the bounds of the address space the *MemHeader* controls; and *mh_Free* tracks the total amount of free memory remaining within that space.

*The mh_First* field, which is a pointer to a *struct MemChunk*, needs a bit more explanation.

Amiga memory allocations always return a block that begins on an even eight-byte boundary, and their size is always an eight-byte multiple. This practice guarantees that every block of free memory is at least eight bytes in size; Exec takes advantage of this to use the first eight bytes of each free block to record information about that block. The information is in the form of a *MemChunk* structure:

```
struct  MemChunk {
   struct MemChunk *mc_Next; /* ptr to next chunk */
   ULONG  mc_Bytes;          /* chunk byte size   */
};
```

Notice that the *MemChunk*s are connected as a singly-threaded list (*not* the usual doubly-threaded Exec *List*) via the *mc_Next* field. The *mc_Bytes* field simply gives the size in bytes of this particular chunk of free memory. The root of the list is back in the *mh_First* field of the *MemHeader* structure; the end of the list is indicated by an *mc_Next* of zero.

We now have an overall picture of Exec's free memory tracking. Starting from *ExecBase*, we see a small chain of *MemHeader* structures, from each of which a (usually much larger) chain of *MemChunk* structures extends into the *MemHeader*'s address space. In a busy machine with expansion memory, there might be three or four *MemHeader*s, each with scores of constantly reshuffling subsidiary *MemChunk*s attached to it.

### Allocating with *AllocMem*

Let's see what happens when we make a typical memory allocation call using the *AllocMem* function:

```
mem = AllocMem(1000L, MEMF_PUBLIC | MEMF_CHIP);
```

Exec's first job in satisfying this call is to riffle through its List of *MemHeader* structures to discover if any of them have attributes matching those requested. In this particular case, it will find what it's looking for on all Amigas in the *MemHeader* describing chip memory.

Having done this, it only remains to be seen if one of the *MemChunk*s attached to the *MemHeader* is big enough to accommodate the thousand-byte allocation desired. To determine if this is so, *AllocMem* passes the header address and the specified size to the *Allocate* routine.

*Allocate* starts at the header's *mh_First* field and chains through the list of *MemChunk*s, looking for an *mc_Size* value greater than or equal to the size requested. If it finds one, it adjusts the *MemChunk* accordingly (or cuts it from the list altogether if it is exactly the right size), updates the header's *mh_Free* field, and returns the address of the formerly free space to *AllocMem*.

### Calling *Allocate* from a program

And thus we come to the point. Notice that *Allocate*, unlike *AllocMem*, does not know or care about *system* structures as such: it does not attempt to distinguish a system-supplied *MemHeader* from one provided by a program. This opens the possibility of using *Allocate* (and its partner *Deallocate*) to manage a private pool of memory, within which many small allocations can be made without fragmenting the system.

The strategy is this: obtain (using *AllocMem*, for instance) an amount of memory adequate for your program's total needs. Initialize a *MemHeader* structure describing the block you have obtained, then manage your smaller allocations within that block using *Allocate* and *Deallocate*. When your program exits, release the memory pool (with *FreeMem*).

Apart from minimizing fragmentation, this strategy will also make your memory allocations execute a little faster, since the calls can be satisfied without wading through the system-wide free memory List. An additional benefit is that – as with Intuition's *AllocRemember/FreeRemember* – you can deallocate all your dynamic storage with a single function call. And the code overhead, as you will see in the *Initialize* function of the listing below, is very modest.

Of course, *Allocate* and *Deallocate* are not appropriate for every program; they are best suited to situations where a relatively large number of relatively small allocations must be performed. In such cases they are well worth keeping in mind among the many allocation options available to the Amiga programmer.

### The *allocate* program

The program listed below has two purposes. First, it demonstrates the proper initialization of a *MemHeader* structure and the use of *Allocate* and *Deallocate* to manage memory governed by that structure. Second, it lets you interactively allocate and free memory within a small pool, and shows you the results of each operation. Since the mechanism is exactly that used by Exec for all the higher-level allocation calls, you can use the program to explore various niceties of memory allocation – how fragmentation happens; the 'coalescing' of adjacent free memory blocks – in a microcosm.

The program and its source will be available on the *Transactor for the Amiga* disk for this issue.

**Listing: allocate.c**

```
/* allocate.c

   This program demonstrates the use of the Exec library Allocate() and
   Deallocate() functions, which allow user management of memory allocation.
   A working area of 416 + sizeof(struct MemHeader) bytes is initially
   allocated with AllocMem(). Subsequent allocations are performed within
```

```
   this area using Allocate().

   In this demonstration, allocations are controlled by the user by typing
   commands at the '>' prompt. The command 'n' (where n is a number),
   allocates n bytes within the working area (if possible). After each
   command, a list of currently free and currently allocated blocks is
   displayed, showing the size of each block and its offset from the base of
   the working area; allocated blocks are labelled with a single letter. The
   command 'c' (where c is a letter) deallocates the block labelled with c
   (if it exists). The command '?' summarizes the usage information given
   here. The command '*' deallocates the working area and exits the program.

   To compile and link this program using Lattice 5.0x, I suggest:

      lc -cf -rr allocate
      blink from lib:c.o allocate.o to allocate lib lib:lcr.lib lib:amiga.lib

   To compile and link with Manx, use:

      cc +L allocate
      ln allocate -lc32

   Execute the program with:

      allocate

   from the CLI. (DON'T use "run allocate".)

      Nick Sullivan, August 7, 1989
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <ctype.h>
#include <stdio.h>

#ifdef   AZTEC_C
#include    <functions.h>
#define     __ARGS(a) ()
#endif

#ifdef   LATTICE_50
#include    <stdlib.h>
#include    <proto/exec.h>
#endif

#define SZ(x)           ((long)sizeof(struct x))

#define STORAGE_AREA    ((long)(8 * 52 + SZ(MemHeader)))
#define NOMEM_MSG       "Out of memory"

struct buf {                /* structure describing one allocated block   */
   char *addr;
   long  size;
};

/* Function declarations/prototypes. The __ARGS() macro allows these
   declarations to work for both ANSI and non-ANSI compilers.
*/
void Initialize __ARGS( (void) );
void CloseStuff __ARGS( (UBYTE *) );
BOOL Dialogue   __ARGS( (void) );
void Report     __ARGS( (void) );
int  cmp        __ARGS( (struct buf *, struct buf *) );
void AddBuf     __ARGS( (long) );
void FreeBuf    __ARGS( (char) );
void Usage      __ARGS( (void) );
void GetInput   __ARGS( (UBYTE *, UBYTE *) );

unsigned char
   *storage;                /* memory pool for user-managed allocations   */
struct MemHeader
   *mh;                     /* structure describing memory pool to system */
struct buf
   buffers[52];             /* table of allocated blocks                  */
long
   hibuf;                   /* count of allocated blocks                  */

void
main (argc, argv)
int argc;
char **argv;
{
   Initialize();                    /* set up memory pool              */
   Usage();                         /* give quick reference of commands */

   while (Dialogue())               /* quits when user enters '*', or   */
      ;                             /* immediately if program is 'run'  */

   CloseStuff(NULL);                /* free memory pool and exit       */
}


/* Initialize

   Allocate the memory pool of 8*52 plus sizeof(struct MemHeader) bytes, and
   fill in a MemHeader structure at the base of the allocated area. (It
   doesn't HAVE to be done this way: the MemHeader could be a static
   structure in the program space, or be allocated separately.)
*/

void
Initialize ()
{
   if (!(storage = AllocMem(STORAGE_AREA, MEMF_PUBLIC)))
      CloseStuff(NOMEM_MSG);

   mh = (struct MemHeader *)storage;

   mh->mh_Lower = (APTR)(mh + 1);
   mh->mh_First = (struct MemChunk *)(mh->mh_Lower);
   mh->mh_Upper = (APTR)(storage + STORAGE_AREA);
   mh->mh_Free  = (ULONG)(STORAGE_AREA - SZ(MemHeader));

   /* Every block of unallocated memory in the Amiga system has a MemChunk
      structure as its first eight bytes (which is why the system always
      rounds up allocation size requests an eight-byte multiple).

      If the entire memory pool is exhausted, mh->mh_First will be NULL;
      otherwise it points to the first in a chain of MemChunks.
   */
   {
   struct MemChunk
      *mc;

      mc = mh->mh_First;

      mc->mc_Next  = NULL;
      mc->mc_Bytes = mh->mh_Free;
   }
}


/* CloseStuff

   Print an error message (if any), deallocate the memory pool, and exit.
*/

void
CloseStuff (txt)
UBYTE *txt;
{
   if (txt)
      fprintf(stderr, "%s\n", txt);

   if (storage)
      FreeMem(storage, STORAGE_AREA);

   exit(0);
}


/* Dialogue

   Show the current state of the memory pool, then read a line from the user
```

```
   and execute it as a command. Return TRUE unless the quit command ('*') is
   issued or EOF is read on input.
*/

BOOL
Dialogue ()
{
UBYTE
   inbuf[256],                  /* buffer for user's command line       */
   c;                           /* first character specifies command    */

   Report();                    /* display current state of memory pool */

   do
   {
      GetInput(inbuf, ">");     /* read input line with '>' prompt      */

      if (feof(stdin))
         return FALSE;

   } while (*inbuf == 0);       /* wait for non-null line               */

   c = *inbuf;                  /* get first character of command       */

   if (isdigit(c))              /* a number - allocate that many bytes  */
      AddBuf(atol(inbuf));

   else if (isalpha(c))         /* a letter - free corresponding block  */
      FreeBuf(c);

   else if (c == '*')           /* quit - return FALSE to caller        */
      return FALSE;

   else                         /* something else (possibly even '?')   */
      Usage();

   return TRUE;                 /* don't quit yet                       */
}


/* Report

   Display the current state of the user-managed memory pool. The list of
   free blocks begins in our MemHeader structure and is maintained by the
   system. The list of allocated blocks is maintained by the program in the
   'buffers' array (for didactic purposes only - it wouldn't be needed in
   an ordinary application).
*/

void
Report ()
{
struct MemChunk
   *mc;                         /* found at start of each free block    */
long
   i;                           /* loop index                           */
short
   freetotal,                   /* total of free memory in pool         */
   alloctotal;                  /* total of user's allocations          */

   printf("Free memory          Allocated memory\n"
          "----------------------------------------\n");

   /* Chain through list of MemChunks starting at MemHeader, and step
      through entries in buffers[], stopping when both are exhausted. Free
      blocks are listed on the left; allocated blocks are listed on the right
      and labelled with a letter so the user can reference them. Totals of
      free and allocated memory are accumulated as we go.

      The block locations are given as offsets within the memory pool, rather
      than as actual memory addresses.
   */
   for (mc = mh->mh_First, freetotal = alloctotal = i = 0;
        mc || i < hibuf;
        i++, mc && (mc = mc->mc_Next))
   {
      /* got a MemChunk? print its stats */

      if (mc)
      {
         printf("%31d bytes at %3d     ",
              mc->mc_Bytes, (long)mc - (long)mh->mh_Lower);

         freetotal += mc->mc_Bytes;
      }

      /* else just tab over */

      else
      {
         printf("                      ");
      }

      /* is there another allocated block in the table? print its stats */

      if (i < hibuf)
      {
         printf("%c) %31d bytes at %3d",
              i >= 26 ? 'A' + i - 26 : 'a' + i,
              buffers[i].size,
              (long)buffers[i].addr - (long)mh->mh_Lower);

         alloctotal += buffers[i].size;
      }

      putchar('\n');
   }

   printf("\nTotal = %3d          Total = %3d\n\n",
         freetotal, alloctotal);
}


/* cmp

   A user function called by qsort(). In this case, we are sorting our table
   of 'buf' structures by address. See compiler documentation for more
   information on using qsort().
*/

int
cmp (a, b)
struct buf *a, *b;
{
   return a->addr - b->addr;
}


/* AddBuf

   Allocate a buffer from our private free memory pool (if possible), record
   the details of the allocation in the buffers[] array, and sort the array.
*/

void
AddBuf (size)
long size;
{
char *
   mem;

   if (!(mem = Allocate(mh, size)))
   {
      printf("Unable to allocate %ld bytes!\n\n", size);
   }
   else
   {
      buffers[hibuf].size = size;
      buffers[hibuf].addr = mem;

      printf("Allocated %ld bytes at offset %d\n\n",
            size, (long)mem - (long)mh->mh_Lower);
```

```
      hibuf++;
      qsort((char *)buffers, (int)hibuf, sizeof(struct buf), cmp);
   }
}


/* FreeBuf

   Free the buffer specified by the user as a letter (if valid). Lower-case
   letters correspond to buffers[] array entries 0-25; upper-case letters
   correspond to entries 26-51. Repack the array.
*/

void
FreeBuf (c)
char c;
{
long
   bufnum;

   if (islower(c))
      bufnum = c - 'a';
   else
      bufnum = c + 26 - 'A';

   if (bufnum >= hibuf)
   {
      printf("Buffer (%c) is not in use!\n\n", c);
   }
   else
   {
      Deallocate(mh, buffers[bufnum].addr, buffers[bufnum].size);

      printf("%ld bytes deallocated at offset %d\n\n",
            buffers[bufnum].size,
            (long)buffers[bufnum].addr - (long)mh->mh_Lower);

      movmem((char *)(buffers + bufnum + 1),
            (char *)(buffers + bufnum),
            (51 - bufnum) * sizeof(struct buf));
      hibuf--;
   }
}


/* Usage

   Display quick usage information.
*/

void
Usage ()
{
   printf("Commands are:\n\n");
   printf("   <num>     Allocate buffer of <num> bytes\n");
   printf("  a-z,A-Z    Deallocate the specified buffer\n");
   printf("     *       Quit\n");
   printf("     ?       Help\n\n");
}


/* GetInput

   Display the given prompt, and read a line of input from standard input.
*/

void
GetInput (buf, prompt)
UBYTE *buf, *prompt;
{
   *buf = 0;

   fputs(prompt, stderr);
   fflush(stderr);
   gets(buf);
}
```


**NEW- F-BASIC™ V2.0 -NEW**

**F-BASIC™ is an Enhanced, Compiled Basic Language System.**

- ★ Easy For Beginners
- ★ SO Fast—Beats C And Other Basics!
- ★ Recursive Subprograms
- ★ Over 350 Powerful Commands
- ★ Easy Access To Amiga Libraries
- ★ Comprehensive Users Manual
- ★ Easy Access to Amiga Windows, Menus, Sound, Events & Animation
- ★ Complete For Serious Programmers
- ★ Direct IFF File Support
- ★ Record Structures And Pointers
- ★ Ultra Fast Floating Point
- ★ Sample Program Disk—Over 100 Examples
- ★ Snobol-Like Pattern Matching

The **F-BASIC™** System Also Has A Source Level Debugger

- ★ Debug **F-BASIC** Programs At The Source Level—So Easy.
- ★ Intuition Interface—Windows For Source, Registers, Memory, Variables.
- ★ Comprehensive Users Manual—Full Documentation.

The **F-BASIC™** Language System 2.0—Only $89.95
**F-BASIC™** System and Complete SLDB—Only $149.95

Available Only From The Manufacturer
SEND CHECK OR MONEY ORDER TO:

**DELPHI NOETIC SYSTEMS, INC.**

Post Office Box 7743
Rapid City, South Dakota 57709-7743
Credit Card or C.O.D. Call (605) 348-0791

F-BASIC and FastCom are registered trademarks of DNS, Inc.
AMIGA is a registered trademark of Commodore/AMIGA Inc.


# Notice to Subscribers

Renew early to avoid missing an issue! Check the expiry volume and issue on your mailing label, and make sure you renew your subscription a few issues in advance. For faster processing of your order, please pay by cheque or money order.

# Notice to Non-Subscribers

Fill in the subscription card in the centre of the magazine and subscribe now! Get the most authoritative technical journal for the Amiga delivered right to your door.

# Pass The Ammo!

## *The Amiga versus other personal computers*

**by Ben Williams**

*Ben Williams is Vice President, Engineering, for Black Belt Systems. He is the author of the* ***Board Master*** *printed circuit board layout software, and the designer of the* ***Amiga Video Transceiver****, both of which are produced by Black Belt Systems. Ben last appeared in Volume 2, issue 3 as the author of an article on colour video hardware design. He can be reached on Compuserve in the AmigaVendor forum (PPN: 76004,1771); his HAM radio call sign is AA7AS.*

This article is a compendium of facts and opinions that contrast the clone systems with the Amiga systems – the first time, to my knowledge, that any kind of a complete comparison has been made between the two types of systems in a magazine. Why? Because it's just not popular for a magazine to bite the clone hand... too much cash involved, even within the Amiga community due to the 'bridge' products and the PC series of Commodore machines.

It is my hope that you will use this article as a means to help bring your arguments together, and take the time to show the clone owners, and potential owners, just what it is they are passing up. This is certainly a case of preaching to the converted, but in fact many Amiga owners do not really know what lurks on the other side, so to speak.

There are really many areas involved in making a comparison like this; some very important, and some not so. Let's begin with the areas where the clone machines *look* like they come out on top.

### Vertical software, specialized hardware and price

Vertical software consists of software written for specific applications, such as a package that can run all the financial activities of a coroner's office. The clones have many more such applications than the Amiga series of computers can offer, because there are over 20 million of them out there and there have been many years for such applications to appear. If the individual you are talking to has a need for such a package, then you *must* talk to them in terms of at least an Amiga 2000 and one or the other of the bridge cards.

There is also a far greater range of specialized hardware for the clones, again, simply because the market is larger. Interestingly enough, much of this hardware simply duplicates (and not all that well, either) some of the functionality that's already built into the Amiga. Even so, you can get virtually anything you want for a clone, assuming you can afford it.

These two issues *would* be very important, if it were not for the availability of the bridge products. The Amiga can, in fact, *be* a clone that can run all that vertical software, as well as plug in those myriad hardware add-ons. This voids both points rather neatly. The Amiga also performs all its 'normal' functions concurrently with the bridgeboards, providing incomparable processing power.

A related issue is price – an Amiga with a bridge is substantially more expensive than a straight clone. In business, this is usually only a factor if the cost doesn't buy you capability, which it certainly does – more than you can get in a clone machine no matter how much you spend.

Now let's get into the meat of things, where the Amiga simply out-features, outperforms, and generally beats the clones back into the primordial dust from whence they came.

### Multitasking

The Amiga OS is a sophisticated multitasking system – it has real time performance, unlike OS/2® and most UNIX® systems. Furthermore, the Amiga system can multitask productively in comparatively small memory – with one Mb of RAM, an Amiga can do wonders. With one Mb, an OS/2 or UNIX system can barely run at all – amazing, but true.

PC owners blithely point at MicroSoft (MS) Windows and say "See? it multitasks!"... but no sir, it doesn't. When Windows activates a word processor for input, your spreadsheet stops working completely – in the Amiga, of course, it will recalculate and be waiting for you, answer in hand, when you go back to that window.

The Amiga's windowing performance is far beyond that of MS Windows, even with the multitasking. This is not only due to the superior operating system, but to the blitter hardware. When a window has to be moved, the system processor has to do only cursory calculations involving boundries of the win-

dows. The blitter does the grunt work of moving and redrawing the actual windows and other display elements.

### Video

The Amiga's video display outshines all the available PCs, including the fruit machines. The Apple Mac II, top of the orchard, offers 256 simultaneous colors at the most – and this is not sufficient for many types of images that the Amiga can display with no problem whatsoever. The top of the line clone displays not only don't meet the video capability that the Amiga has, but they can cost more than the entire Amiga system all by themselves!

The sprite capability that the Amiga has is unknown in the clone world – pointers and so on are always drawn onto the screen by the processor, taking valuable time and processing power.

The Amiga is capable of operating, and typically does so in more than one graphics mode at one time – not so a clone. This is immensely useful in a multitasking environment, but it's difficult to explain to a clone user. Although you *can* hook up several monitors to a monochrome adapter and a VGA (video graphics adaptor), for instance, under the operating system only one may be operational at one time, so there is little point in going to that expense.

The Amiga's video is also compatible with VCRs and televisions, something clone and Apple owners dream about, or spend thousands of dollars to achieve. Desktop video may be overhyped, but in fact the Amiga is the only machine that can play that game without an expenditure that would buy you an Amiga 2500 and leave you cash for several megs of 32-bit memory.

### Sound

The clone machines have a one bit control line that is used to drive a speaker. This results in razzy, 'R2D2-like' beeps and bops. How exciting.

The Amiga provides full stereo output at common line levels, which means it connects right to your stereo; and the sound quality is such that you will want to. There are four separate internal channels (two per stereo channel) that are used to produce the Amiga's sound output, and these are driven by more custom hardware in the machine – not by the processor, which means that the sound can be much more sophisticated. This is because the processor is able to concern itself with the characteristics of the sound, rather than the dumb brute movement of vast quantities of data to the sound outputs. It also means that the Amiga can make complex, continuous sound while it multitasks – without degradation. Try *that* on another machine!

You might also consider the built-in speech that, although it wouldn't win any toastmaster's awards, certainly offers major utility to those who cannot watch the monitor for any reason, from process control monitoring to physical blindness. No matter how you look at it, it's certainly no drawback.

### Standard I/O Capabilities

The Amiga comes with a serial and a parallel port – clones may or may not have these, so the hardware part of it is sort of a non-issue. But what the Amiga can do with those ports is another thing entirely...

Because of the highly efficient multitasking OS, the Amiga doesn't need things like multihundred-dollar printer buffer add-ons. A clone must finish printing to a printer or an expensive external intermediate storage device before it can do useful work again. Some programs, such as Word Perfect, do provide an internal spooler for you. However, you cannot exit WordPerfect until the spooler has terminated. Furthermore, many such programs do not co-exist well with each other's versions of spoolers.

The serial port capabilities are similarly enhanced by the Amiga OS. Being online to a network, a salesman in the field, or a friend is no impediment to other activity on the machine. A clone is usually completely occupied by such an activity – again, a terrific impediment to productivity.

You can also add boards to an Amiga 2000 or 2500, such as ASDG's excellent dual serial card, and actually run more than one serial application at one time. I often run my Terminal Node Controller on the amateur packet network while I'm online to Compuserve, and of course I'm free to wander thru database, CAD or word processing software at the same time.

*The Amiga system can multitask productively in comparatively small memory – with one Mb of RAM, an Amiga can do wonders. With one Mb, an OS/2 or UNIX system can barely run at all.*

The availablity of two multifunction controller ports provides constant availability of a mouse (which is standard, of course) without tying up the serial port or preventing the attachment of a joystick. The Amiga can handle mice, switch-based joysticks, linear joysticks and light pens with no problem – standard.

### User interface

The clones are generally stuck in the typist's world; the Apple machines are stuck just as hard in the iconic world. The Amiga handles both interfaces, and it does them well, too. The CLI offers an environment similar to, but much more powerful than, a clone. The Workbench offers an environment similar to, but much more powerful than MS Windows. The Apple machines have a more polished Workbench style interface, although this may not be true after the next Amiga operating system upgrade.

## Base Horsepower

Commodore offers Amigas with processors currently ranging from 7.14-MHz 68000's, approximately the equal of an Intel 10-MHz 80286, to 14-MHz 68020's, easily comparable to 20-MHz 80386 machines in base hp. You can also get 68030 processors running at 25 MHz or more from 3rd party vendors as I write this.

But, and this is very important, the speed comparison between the 68xxx series and the 80xxx series is only valid if the processors are used in their native modes. The fact is, 80286 and 80386 processors are used almost uniformly in the clone world to emulate an 8088, working with a 640k memory limitation, and completely without the benefit of the advanced instructions that these processors offer. The net result is that the base Amiga, using only a 68000, can out-power even a 20-Mhz 80386 in many applications.

How? The 68000 series of processors, and the Amiga, offer a linear addressing architecture. This means that the maximum amount of memory that can be handled by the processor is much, much larger than that of the 8088 and 80286. The 80386 can address more memory than it's crippled companions, but as pointed out previously, is unable to do so when emulating an 8088/86 – in other words, if it's running MS-DOS.

If a clone needs to address a large data array, such as one Mb of statistical samples, tough luck. Thrash it on and off of disk (really slow) or keep it in the EMS (Extended Memory Standard) extended memory card, where the processor has to go through a three ring circus to manipulate the data – also slow.

The Amiga, by contrast, can have many megabytes of data in memory at one time, along with multiple programs and miscellaneous system structures. The Amiga does this without any performance degradation at all. As a result, you see programs on the Amiga that are more complex and more capable than those on the clone machines – and the potential is there for even greater gains.

The Apple machines share this architectural advantage, but do not have the coprocessing power to take as much advantage of it as the Amiga can, nor a powerful OS to complement the architecture.

The power of the internal hardware architecture of the Amiga only begins with the 68xxx processors. Standard with the system are three coprocessors which manage sound, disk, graphics, the mouse, linear joysticks, light pens, and 32 DMA channels. If one looks at the Amiga 2500, you can add to that the 68881 math coprocessor, which is similar to the math coprocessors optionally available for some (but not all) clone machines; and a memory management unit that can do useful things with the 2 megabytes of 32-bit ram contained in the A2500 system in addition to the standard one megabyte the A2000 has.

These things, when taken together, create a situation where a clone cannot possibly compete with the Amiga in any area other than cheapest price. If you consider the inclusion of a bridge card a necessity, which it currently is if the machine is to be used in a majority of business situations, then you must pick at least an Amiga 2000. The price of a A2000 plus the bridge is three to four times the cost of a basic business capable clone, and if the issue is price, forget it. If the issue is performance, then it is clear that the purchase of any clone machine is a major mistake, no matter who the purchaser is, or which machine is at issue.

## Now for the bad news

There is a serious problem lurking in the Amiga universe. It is one which would have destroyed the machine before it ever got started, were it not for the thousands of visionary individuals who have supported the machine since the beginning.

The problem is marketing – or, I suppose I should say, the lack of it. After four years of crippled or completly absent marketing efforts within the United States, the machine remains virtually unknown. The Amiga's worst enemy remains the company that sells it. After all this time, I despair of those who count at Commodore actually getting aggressive and *selling* the machine.

There is absolutely no indication that they know of the Amiga's marketability, or even it's superiority. An ad in a clone publication that states the simple fact that purchase of any machine other than an Amiga is an error of a magnitude that should get people fired is the kind of thing I'm talking about.

The fact that the Amiga remains a viable competitor in a climate like this is only one more indication that it is by far the best machine you can buy – no matter what you spend.


**AMIGA**
**COMMODORE**
CHIPS...PARTS...UPGRADES

| | | | |
|---|---|---|---|
| 6526 | $12.25 | 8364 (PAULA) | $56.95 |
| 6567 | 15.95 | 8362 (DENISE) | 56.95 |
| 6510 | 10.95 | 8370 (FAT AGNUS) | 59.85 |
| 6581 | 11.95 | 8386 (GARY/5719) | 17.25 |
| PLA (825100) | 12.95 | 8520 AI | 17.95 |
| 901 ROMS | 11.25 | A501 UPGRADE RAM | 139.50 |
| 8562 | 22.50 | 1.3 KICKSTART ROM | 27.95 |

**MANY OTHERS IN STOCK**

**• JUST RELEASED •**

**AMIGA UPGRADE...**New 1 MEGABYTE "FATTER" Agnus Chip (#8372) allows users more "Chip" memory for use in GRAPHICS, MUSIC and VIDEO. This upgrade is an absolute must for present AMIGA owners. Price is $113.50 with instructions.

**COMMODORE DIAGNOSTICIAN II**

JUST OUT – a newly revised/updated version of the COMMODORE DIAGNOSTICIAN which sold over 10,000 copies world-wide. The COMMODORE DIAGNOSTICIAN II locates faulty chips on all COMMODORE computers/1541 drives and has different sections such as "cross reference #s". This diagnostic tool had a fantastic full page review in last years "Computer Shopper Magazine". Cost is $6.95 prepaid to North America.

We also sell RAM/SIMMS testers, diagnostic evaluators, cables and other exclusive new products for Commodore and IBM.

**Send for catalog on exclusive new products**

**The Grapevine Group, Inc.**
**35 Charlotte Drive**
**Wesley Hills, N.Y. 10977**
**1-800-292-7445** **FAX 914-354-6696**
**914-354-4448**

MasterCard
Dealer Prices Available
Price subject to change
VISA

# Amiga Disk Encoding Schemes

## *MFM? GCR? Please explain!*

**by Betty Clay**

*Betty Clay, a former mathematics teacher, chose writing for her second career. She is the Amiga columnist on STARTEXT (a videotext service of the Ft. Worth Star-Telegram). She has a regualr column with ICPUG, and her work has also appeared in The Amigan. Betty can be reached via CompuServe (PPN: 76702,337), QuantumLink (bjc) or by mail at 1605 Glasgow Drive, Arlington, TX. 76015.*

Disk drives are fascinating things. When they work properly, they manage data and software so beautifully that there is little need to think about the way in which they do it. When something goes wrong though, the loss is (or can be) so great that the inner workings of the drive become of utmost importance.

When I first began to ask about how disk drives work, I was usually told that drives were intended to be black boxes and that I was not supposed to understand them – just use them with gratitude. (Was someone as ignorant as I, or was he being a bit sexist?)

Eventually, a disk editor entered my life, and I learned to read and interpret the hexadecimal representations of what was on my disks. Poring over the disk manuals provided some information, and I learned to read directory tracks, find where things were actually located on my disk, and sometimes rescue things that would otherwise have been lost. It was through the writings and teaching of Mike Todd in England, and of Dick Immers and Gerry Neufeld in North America, that I learned about disk encoding schemes, headers, header gaps and all the other niceties that make up the *real* layout of the disk. It's not that I ever actually saw a disk block encoded in GCR, but I did know what was written there, how it was written, and how it was read back.

One of my first projects was to work out the disk layout for the then new Amiga. It was a worthy task, because corrupted disks were very common in those days, and we did not have DiskSalv to rescue them.

Among the very few utilities available in those earliest days was an editor called DiskEd. Perhaps I was the only person who really liked DiskEd, but it enabled me to learn much of the disk structure early on. There are ways in which I find it superior to any of the later, more popular, editors. With DiskEd, and the sparse information in the *ROM Kernal Manual*, I was able to learn about the header information within each block and the hash tables used in directory blocks, and to follow a file from the root to its actual location on the disk.

But for me, this was still not enough. I was reading a representation of what was on the disk, but I still did not know what was really there.

The Rom Kernal Manual told us from the first that the Amiga drives were capable of writing in either GCR or MFM format. It did not give a further description of either. GCR was familiar because it is used in the other non-Amiga Commodore drives.

MFM, while an old method of writing on disks, was new to me. I went about seeking information on this method, but found little. Everywhere I turn, people say that "this is an MFM drive" or "this is a GCR drive" or "this is a(n) xxxx drive", but no one could ever explain exactly what these designations meant. Days of searching through libraries have been fruitless. There seem to be people who *do* know it, and they use the terms freely, but I could find very few references in print. The information that I have discovered has been in tiny bits and pieces. Put together, those tidbits seem to be sensible and correct. Here is what I have learned.

### Why encode?

Within the computer, information is stored as on-bits and off-bits (represented most often by zeros and ones), but it decodes them so that we see alphanumeric characters. Similarly, the data on the disk can be written as a string of zeros and ones – or more accurately, as a string of magnetized bits of oxide.

The read/write head on a disk drive is very much like a coil. As current passes through this coil, it creates a magnetic field, which is recorded onto the magnetic disk. If the current goes through in one direction, the bit is recorded with a 'north pole'; if another way, with a 'south pole'. If nothing at all is recorded, the disk will contain a null at that spot. Nulls occur only in the gaps between data; the Amiga places a group of nulls at the end of the series of logical sectors on a track. Some kinds of drives place nulls between individual sectors, which

allows the reading of a single sector from a track, rather than in whole tracks as done on the Amiga.

When the read-head is passing over a track, it ignores the nulls in the gap. If it passes over a magnetic field, a pulse is generated, and is interpreted as data. Since there are no nulls within sectors, the presence of a pulse can be interpreted as an on- or one-bit; the absence of a pulse as an off- or zero-bit.

The disk spins in the drive at a constant and carefully controlled speed of 300 rpm. Amiga drives are constructed to always be within 1.5% of this speed. This allows about .2 seconds for the reading of a whole track of data, including eleven sectors and their headers. The bits must be written to the disk such that the drive can accurately determine whether each bit is a zero or a one. Physical restraints of the drive require that there be at least two microseconds between polarity changes, or 1-bits.

In order to make reading more accurate, encoding schemes have been developed that make it impossible for two 1-bits to occur too close together for accurate reading. They also establish rules for writing data, making it possible to write some bits that could not possibly *be* data. These schemes serve two purposes. The drive must be able to determine where the data starts if it is to interpret the binary digits accurately, and it also must be able to accurately determine whether any given bit is zero or one.

The start of data is marked on the disk with a synchronization mark – a pattern of bits that could not possibly occur in normal data. On the 8-bit Commodore drives, the sync mark consists of ten or more 1-bits in a row, since normal data is not allowed to have more than eight consecutive ones. On the Amiga, a data bit can never have more than four zeros, and a sync mark. Bits of data are encoded in ways that prevent too many on-bits or too many off-bits in a row, so that the drive can accurately read them. While there are many methods of encoding the data, the two mentioned here are the two found in the Amiga, GCR and MFM.

## GCR encoding

There are several forms of GCR encoding, but the one described here is the one used in the 8-bit Commodore drives. The Amiga is capable of using GCR, but no one seems ever to use it. There is one apparent advantage to GCR – more data can be stored on a disk encoded with GCR than with the MFM we normally use.

GCR stands for Group Code Recording, perhaps because the bits are grouped together before encoding. In order to use this system, each byte is separated into two four-bit nybbles. Each nybble is then encoded according to a GCR table, with each four-bit nybble being encoded as five GCR bits. Using GCR, it can be determined that there will never be more than eight consecutive one-bits or two consecutive zero-bits. The laws of probability tell us that for *n* distinct members of a set, there can be two to the *n* power possible arrangements of them. Thus, for groups of four bits, there can be exactly sixteen ways in which they might be encoded. The GCR table, therefore, consists of sixteen translations:

| Decimal | Hexadecimal | Binary | GCR equivalent |
|---|---|---|---|
| 0 | $0 | 0000 | 01010 |
| 1 | $1 | 0001 | 01011 |
| 2 | $2 | 0010 | 10010 |
| 3 | $3 | 0011 | 10011 |
| 4 | $4 | 0100 | 01110 |
| 5 | $5 | 0101 | 01111 |
| 6 | $6 | 0110 | 10110 |
| 7 | $7 | 0111 | 10111 |
| 8 | $8 | 1000 | 01001 |
| 9 | $9 | 1001 | 11001 |
| 10 | $A | 1010 | 11010 |
| 11 | $B | 1011 | 11011 |
| 12 | $C | 1100 | 01101 |
| 13 | $D | 1101 | 11101 |
| 14 | $E | 1110 | 11110 |
| 15 | $F | 1111 | 10101 |

When a byte of data is to be encoded, the process goes something like this:

Take a byte of data, say the number 18, which is $12.

| | |
|---|---|
| Binary notation: | 0001 0010 |
| Convert the high nybble to GCR: | 01011 |
| Convert the low nybble to GCR: | 10010 |
| And the concatenated byte is now: | 0101110010. |

Note that there are never more than two zeros in a row. No item in the lookup table consists of all ones, and an examination of the table shows that there can never be more than eight consecutive ones.

Since GCR bytes contain 10 bits, an 8-bit computer byte always leaves some bits left over after filling the allotted space. For this reason, it is customary for a GCR drive to handle four 10-bit bytes at a time. This gives a total of 40 GCR bits, which can be handled in the same manner as five regular bytes. The number is chosen because 40 is evenly divisible for translation of 8-bit bytes, being the least common multiple of eight and five.

Since four binary bits are encoded as five GCR bits, there is 80% efficiency. That is, 80% of the available bits can be used either for your own data or for overhead information needed by the drive.

This would appear to let a disk hold far more information than could be stored under most other methods. However, since GCR permits the use of as many as eight on-bits in a row, the drive cannot interpret them at full speed. It is necessary to write or read at only half the normal speed, in order to insure accuracy. When the writing speed is slowed to four microseconds per bit instead of the normal two, the density of the data is only half as much, cutting drastically into the storage advantage. On older drives for

both the Apple and Commodore machines, increased capacity was more important than speed, so GCR was used by both. The GCR in the Amiga is actually compatible with the Apple version. The lookup tables are the same for both, but the actual implementation is a bit different.

I once asked Jay Miner why the GCR table and capability were built into the Amiga but never used. He said that they wanted it for compatibility reasons, but preferred the faster and easier MFM encoding for normal use. Should you ever need it, it is sitting there waiting for you, and it does not take up very much room.

### MFM encoding

This is the encoding method currently in use on the Amiga floppies – and on most of the hard drives as well. This format puts both data bits and clock bits onto the disk. Since there is one clock bit recorded for each data bit, only 50% of the bits on the disk represent information that is needed. The clock bits are added as the data is written out, and are deleted as it comes back in.

The system does not require a lookup table, because the encoding rules are very simple. If either of the two adjacent data bits is a 1-bit, a zero is inserted between them. If two adjacent data bits are both zeros, a one is inserted between them. With this system, it is certain that two 1-bits will never be adjacent to each other. The system can thus read and write at two microseconds per bit as opposed to four microseconds for GCR.

Let us take that number 18 again:

```
Binary notation:      1   0   0   1   0
Get the clock bits:  0   0   1   0   0
So, encoded, it is:  0 1 0 0 1 0 0 1 0 0
```

Timing also determines the interpretation of these pulse/no-pulse bits. The computer knows it is supposed to be passing a new bit every two microseconds, so it interprets the data as having one zero each two microseconds if it does not determine a change of magnetism in that time.

When data is written to the disk, an entire track is written at one time, but it is managed as though it were done in sectors. Each sector has a header of sixteen bytes, but this header is not seen by normal users of the drive. According to the *ROM Kernal Manual*, the Amiga sector headers, before encoding, contain:

- two bytes of $00 data, represented in MFM by $AAAA
- two bytes of $A1, a standard MFM sync byte, which is $A1 without a clock bit
- one byte to tell what format (for *1.0, this format type was $FF)
- one byte to hold the track number
- one byte to hold the sector number
- one byte to hold the number of sectors to the end of the write
- sixteen bytes of operating system recovery information
- four bytes of header checksum
- four bytes of data checksum and this is followed by the 512 bytes in the sector that we are able to read with a disk editor.

The encoding of these bytes is done in a curious way. The first four bytes of the header are encoded one byte at a time. First, the odd bits are separated from the even bits. The odd bits are then encoded, following the rules for MFM, and are written out to the disk. Next the even bits are shifted left one position, and then they are encoded and written out.

Using our number 18, should it be in one of those bytes, it would be handled thus:

```
18 = $12     hexadecimal
   = 10010   binary
     1 0 0   for the odd bits
    0 0 1    for the odd bits clock-bits
    010010   written out to the disk
      0 1    for the even bits
    0 1      shifted left one position
   1 0       for the clock bits
   1001      written out to the disk
```

So that the entire representation of the number 18 on the disk would be `0100101001`.

This would be done for each of the four bytes of header and sync marks. The next four bytes (format, track, sector, and number of sectors to end) are all encoded in a single block, following the same pattern: separate the odd bits from the even, encode the odd bits and write them out, encode the even bits and write them out. The 16 bytes of operating system recovery information are then encoded as a single block, in the same manner. Then the four bytes of header checksum are handled as a single block, followed by the four bytes of data checksum as another unit. After this, the entire data sector of 512 bytes is encoded as a single unit: first the odd bits, then the even ones.

All encoding is done in the blitter. The data is read from the disk, decoded in the blitter, and placed in a user buffer. What you see in your buffer will always be decoded. It is probably possible to bypass the blitter and read the actual encoded information, but I do not know of anyone who has done so.

### Blocks to end of write

Perhaps this is not really a part of the encoding process, but the 'blocks to end of write' is a curious feature of AmigaDOS.

When you write to an Amiga floppy, the drive heads are given a track number to which they can write. They then move immediately to the assigned track and begin writing. They do not wait for a sync mark, nor do they look for a particular sector, and they are not likely to write on the same part of the track that was used before.

Remember that sectors on the Amiga are logical, not physical. There are no null gaps between sectors, though there is a rather large gap of nulls at the end of the last sector on the track. The heads just start writing at the spot on which they have found the track. If this is the very first time that track has

been written to, the sectors on the track will be written in consecutive order. The heads will write a group of null bytes, and then write out the sectors. In the sector headers, the 'sectors to end of write' (the offset) would be like this:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sector Number: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Offset: | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

This offset number tells the drive how many more sectors there will be before the gap of nulls. When this track is read back, the heads will again begin to read as soon as they find the correct track, not waiting to look for any kind of marking. As these sectors are read, decoded, and put into the user buffer, these offsets go along with them.

Suppose that when the head began to read, it was over sector five. Then the read would be much like this:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sector Number: | 5 | 6 | 7 | 8 | 9 | 10 | 11 | nulls | 1 | 2 | 3 | 4 | 5 |
| Offset | 7 | 6 | 5 | 4 | 3 | 2 | 1 | | 11 | 10 | 9 | 8 | 7 |

Notice that sector five must be read twice, because it is not likely that the first read began at the beginning of the sector. In practice, the first read of this sector would be discarded for that very reason. In the user buffer, the information would then be moved, but not as you might expect. Sector six will be moved to the beginning of the buffer, followed by the others through sector eleven. The nulls are discarded, and sectors one to five will follow sector eleven.

When this information is written back to the disk, the sector offsets will be changed to reflect the new order of the sectors:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sector Number: | 6 | 7 | 8 | 9 | 10 | 11 | 1 | 2 | 3 | 4 | 5 nulls |
| Offset | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Notice that there is never a zero in the offset byte because this offset must be read before the sector is read, and there is always at least one more to read before the nulls at the time the offset is checked. This is quite important because the track could end with any sector, and the drive must know when to stop reading.

The sixteen bytes of operating system information are not normally used. The user of the disk will never see the information here, but it is possible for programmers to store data there, so long as the space is not required for some future revision of the operating system. It is not suitable for copy protection, but it might be used for storing copyright information, serial numbers, and the like. Your imagination is required here!

The placement of the nulls is the reason why AmigaDOS disks appear to be good upon first format, yet are reported as bad disks upon later writes. The earlier accesses wrote to good parts of the track; the bad parts landed in the nulls that follow the last data sector, so the disk was reported to be good. On a later write to the disk, the information begins at a different place on the track, so the bad part will no longer be in the gap of nulls.

## The next frontier

The Amiga drives still hold quite a few mysteries. Perhaps they will all be revealed in time, perhaps not. The entire subject of hard disks is yet to be explored. Those black boxes yield their secrets slowly, but with persistence, they *can* be discovered.

**Further reading:**


Clay, Betty, "More on Amiga File Structure", *Transactor*, Volume 7, Issue 06, May, 1987, pp. 72-73.

Grote, Gelfland, Abraham, *Amiga Disk Drives Inside and Out*, Abacus, 5370 52nd Street SE, Grand Rapids, MI 49508, first edition (1988).

Immers, Richard and Neufeld, Gerald, *Inside Commodore DOS*, Datamost, 20660 Nordhoff Street, Chatsworth, CA 91311-6152, (1984).

Peck, Robert A., Sassenrath, Carl, and Deyl, Susan, *Amiga ROM Kernal Manual*, Volumes I and II, Commodore-Amiga, Inc., (1985).

Sterling, Thom, "Fun with the Amiga Disk Controller", *Amazing Computing*, Vol. I, No. 6, July, 1986, pp 73-74. Pim Publications, Inc., P. O. Bos 869, Fall River, MA. 02722.



**GAME PROGRAMS WANTED!**

Attention Developers and Freelance Programmers!

Virtual Reality Laboratories, Inc. is looking for a few sharp new entertainment game and educational game programs for the Amiga. To be accepted, these programs must be highly graphic with excellent sound capability. For a rapid review of your work send for guidelines to VRLI, Box 609, Atwood, CA 92601.

Do not submit until you have received and read our guidelines, but please do send for the guidelines TODAY!

# IMG Scan

## *Scanning at an affordable price*

### Review by Paul Bosacki

**Image scanning hardware/software**

**from SunRize Industries, PO Box 1453, College Station, TX 77841, USA.**

**Suggested Retail: $149.95**

*IMG Scan* package contains image scanning hardware and software for the the Amiga 500/2000 series. The hardware consists of a small interface box the plugs directly into the parallel port of the Amiga. The parallel port is passed through to the other side of the box.

From the interface extrude two three-foot lengths of fibre-optic cable encased at the other end in a short metal tube. The metal tube forms the head of the scanner. This is attached to the print-head of any dot-matrix printer capable of performing variable-spaced vertical printing. A spring clip is used to mount the scanhead.

The hardware, and the theory behind it, is amazingly simple. An infrared emitter and detector are teamed with an 8-bit A/D converter. On a scan, darker areas absorb more of the emitted

infrared energy, while lighter areas reflect more of that energy back to the detector. The result: the voltage output of the infrared detector is higher for those lighter areas. The A/D converter converts that difference into a number between 0 (lightest) to 255 (darkest). Consequently, the IMG Scan hardware is capable of interpreting 256 gray levels. This does not, however, limit the user to low-quality scans. The scanned images (lower left) are quite impressive when the cost of the product and its inherent short-comings (the consquence of ambient light, for example) are brought into perspective.

This is not, however, an unqualified acclaimation of the product. The software is lacking in some respects. And only some trial and error will get you the scan you're looking for.

The software offers four screen resolutions: 320x200, 320x400, 640x200 and 640x400. The manual suggests that all scans be initiated from the 320x200 screen. The screen type can be changed afterward with a corresponding increase (at higher resolutions) in the quality of the scan.

The scan areais set through modifing parameters from the keyboard. The problem here is the difficulty relating those numbers to the actual scan area. The default horizontal offset is 500. Decreasing this value will set the scan area closer to the left-hand margin, but by how much is left to trial and error. If you have erred, pressing the space bar will terminate the scan.

Another fault lies in setting the vertical scan area. This is done by modifying the horizontal spacing. For example, a horizontal spacing of 4000, sets the scanned area to four vertical inches. Whether the scanned area is four inches or eight inches though, the scanhead makes the same number of passes. Consequently, the aspect ratio when compared to the original will suffer.

In some senses, the user interface is poorly designed. A case in point is the *Edit* box. Missing are the front-to-back gadets. Called up from the *Edit* box, a histogram appears behind the box. To view the histogram, the *Edit* box must be closed. To get rid of the histogram, however, it is necessary to recall the *Edit* box and click on the *Update* gadget to redraw the screen. Just not as easy as it could have been.

With that said though, I am impressed with IMG Scan and SunRize Industries. The utility of the package, and the inexpensive scanning it offers, commends itself to the occassional desktop publisher. This product falls just short of offering professional results. And that, in itself, is high praise.

# NewsLink

## Amiga talks MS-DOS

**From Amiga to Big Blue**: Consultron releases *CrossDos*, software that allows the user to read and write 360K and 720K MS-DOS/PC-DOS/Atari ST formatted disks directly from the AmigaDOS environment.

Because CrossDos mounts an MS-DOS filesystem, it fully integrates itself into the Amiga operating system allowing MS-DOS formatted disks to be "transparently" accessed from virtually any utility or application, just as any other Amiga file or directory.

Most AmigaDOS command line functions are available with CrossDos:

- relabel a disk
- create directories
- rename files/directories
- set file/directory dates
- set/clear file/dir attributes
- seek file positions
- obtain disk information
- allocate more cache buffers.

CrossDos offers full compatiblity with the WorkBench, two ASCII text filter options and format and copy utilites. It comes with a 'kill' utility to recover memory after use. CrossDos comes complete with an installation program.

CrossDos is suitable for those users who don't need the power of a bridge card but still need to get files back and forth.

CrossDos lists for $30.00 US. A 'read-only' version is available in the public domain or from Consultron themselves for $5.00 US.

**Contact**: Consultron, 11280 Parkview, Plymouth, MI 48170, USA.

*Because CrossDos mounts an MS-DOS filesystem, it fully integrates itself into the Amiga operating system...*

## ...and Macintosh

**Once you go Amiga, you'll never go Mac, but...**: Central Coast Software introduces *Mac-2-Dos*, a combination of hardware and software that allows the transferring of Macintosh files to and from the Amiga. A standard Macintosh-compatible 3.5-inch external floppy disk drive is required.

Mac-2-Dos converts *MacPaint* files to and from Amiga ASCII. It also transfers *MacBinary* and *PostScript*. Other conversion options include resolution choices which simplify the problems caused by the incompatible screen resolutions.

Mac-2-Dos provides a split-screen directory listing of Amiga and Mac files available for transfer. You can select individual files for transfer, 'tag' multiple files, or transfer groups of files with one command.

Mac-2-Dos includes a custom hardware interface, driver software, file conversion software, and optionally, a Mac-compatible 3.5-inch floppy drive. The hardware interface plugs into the Amiga's external disk drive connector, or into the last external drive of the chain.

Mac-2-Dos works with any Amiga, supports multitasking, and conforms fully with the software and hardware guidelines for the Amiga.

If you already own a Mac-compatible drive, you can use it with the Mac-2-Dos interface on your Amiga to transfer Mac files. If you purchase the Mac drive with Mac-2 Dos, that same drive can be used to read and write standard AmigaDOS disks.

Two packages are available. Without a Mac-compatible drive, but with the necessary hardware and interface software, Mac-2-Dos retails for $99.95 US. The same package with the Mac drive costs $349.95 US

**Contact**: Central Coast Software, 424 Vista Avenue, Golden, CO 80401, USA.

*If you purchase the Mac drive with Mac-2 Dos, that same drive can be used to read and write standard AmigaDOS disks...*

## Parallel Processing Products

**Parallel Processing for the Amiga**: Digital Animation Productions has released a number of products which bring the technology of parallel processing to the Amiga 2000. The *SANG Transputer Board* for the Commodore Amiga 2000 requires an 8-bit PC slot and a Zorro 16-bit slot.

The basic transputer configuration includes one INMOS T800 (a 32-bit RISC parallel architecture transputer), and 1 Mb of RAM. The unit can be expanded to four INMOS T800s and 16 Mb of RAM.

Each T800 includes four kbytes of fast internal RAM. This permits fast access of frequently used data and code without the waits states imposed by accessing slower, local memory.

On chip floating point units operate in parallel to the main processor offering up to three million floating point operations per second. T800 chips are available at operating speeds of 20 MHz, 25 MHz and 30 MHz.

Each transputer individually provides for the capacity of parallel processing. On a fully configured board containing four transputers, each transputer has one link routed directly to each of its three counterparts and another link routed to an external connector.

These links provide high-speed communication channels (typical data rates of 20 megabits per second) running independently of the processor and the floating point unit.

Some support hardware and software is available. Among these are a video graphics transputer, *SANG Parallel Memory Board* and the *INMOS B004 Proto-typing Board.*

Software developments include: *Parallel C*, *Parallel Pascal*, *TASM 2*, *OCCAM Compiler/Server OCS* and the *HELIOS OS*.

**Contact**: Digital Animation Productions, 79 Mt. Vernon St., Boston, MA 02108, USA.

*The basic transputer configuration includes one INMOS T800 (a 32-bit RISC parallel architecture transputer), and 1 Mb of RAM. The unit can be expanded to four INMOS T800s and 16 Mb of RAM...*

## Versitile Video Interfacing

**CMI does video adaptors**: Creative Micro systems introduces the *VI-Series Multiple Output Video Adaptor* for the Amiga 500 and 2000. It allows the Amiga to be connected to virtually any video equipment. There is no specialized software involved; a few cable connections or the installation of an internal board is all that is needed to interface the Amiga with a wide variety of video hardware products.

The VI-Series adaptors provide video output in standard formats. With the exception of genlock devices, only the VI-Series provides the quality video output signals necessary to perform desktop video with the Amiga.

CMI VI-Series adaptors allow transmission directly from the Amiga to the following types of video hardware: virtually all composite monitors, Commodore 64 and 128 monitors, VHS and Beta VCRs. With the addtion of a separate cable, VI-Series adaptors can also support the near-broadcast quality of Super VHS.

*CMI VI-Series adaptors allow transmission directly from the Amiga to the following types of video hardware: virtually all composite monitors, Commodore 64 and 128 monitors, VHS and Beta VCRS...*

The VI-Series adaptors are available in four formats. The *VI-500 Adaptor* is an external device designed to send a video signal from the Amiga 500, 1000 or 2000 to various video hardware. It is enclosed in a metal case, and attaches to the RGB port of any Amiga via a 23-pin connector.

Three RCA jacks providing COMPOSITE OUT, CHROMA OUT, and LUMA OUT allow for interfacing with Commodore and composite monitors, VHS and Beta VCRs and Super VHS hardware with an additional cable.

The *VI-500RF* is identical to above except for an additional RF OUT co-axial cable jack. Both the *VI-2000* and *VI-200RF* are internally mounted boards for the Amiga 2000. They offer the same outputs as the VI-500X series adaptors. Costs are as follows: VI-500 or VI-2000, $79.95 US; VI-500RF or VI-2000RF, $99.95 US.

**Contact**: Creative Microsystems Inc., 19552 SW 90th Court, Tualatin, OR 97062, USA.

## CMI Accelerates Amiga

**CMI Accelerates the Amiga**: Want to double processor speeds? CMI offers a processor accelerator for the Commodore Amiga series. Their *MC68000 Processor Accelerator Board* sports the following hardware:

- 16 MHz SGS-Thompson 68000
- Built-in math co-processor socket
- Built-in fast ROM feature
- Built-in Commodore compatible Kick Start ROM socket (Amiga 1000 version)
- Optional battery-backed clock (Amiga 1000 version).

The accelerator board simply attaches to the MC68000 socket on the Amiga motherboard replacing the orginal 68000.

Because of the advanced 16 MHz SGS-Thompson 68000 chip, Amiga instruction speeds are increased from 7.16 MHz to 14.32 MHz. This amounts to doubling of microprocessor speeds.

With CMI's fast ROM access, all KickStart routines are accessed at 14.32 MHz as well. Depending on the nature of application, expect system throughput to increase by as much as 40%.

Adding a math co-processor, either a MC68881 or MC68882, will increase processor speeds even further with those applications that are compiled to access the IEEE libraries of WorkBench 1.3.

Of course, the MC68000 micoprocessor is limited to treating the the MC6888x FPU as a peripheral device. This is a limitation of the microprocessor, and not the CMI unit.

CMI's MC68000 Processor Accelerator Board for the A1000 includes an optional battery-backed clock. A Commodore-style KickStart ROM socket comes standard. To have KickStart in ROM on the A1000 simply purchase the 1.3 upgrade ROM and install it on the accelerator board.

With that done, 256k of system memory can be freed up for use. This requires modification to the to the Amiga daughter board that is explained in the manual.

CMI offers versions for the A500, A2000 and A1000. The Amiga 500 or 2000 *Processor Accelerator Board* is available for $250.00 US. The accelerator for the Amiga 1000 lists at: $280.00 US.

**Contact**: Creative Microsystems Inc., 19552 SW 90th Court, Tualatin, OR 97062, USA.

*With fast ROM access, all KickStart routines are accessed at 14.32 MHz as well. Depending on the application, system throughput can increase by as much as 40%...*


pro · duc · tiv · i · ty

*n.* The quality or state of being productive.
ie: 1. producing abundantly. 2. marked by abundant production: as, a *productive* time.
3. as in *M2Sprint Development System.*

Amiga & ***M2Sprint*** Development System with Programmer's and Library Ref. Manuals and six release disks

**M2Sprint**

The ***complete*** **Modula-2** development environment for the Amiga that makes programming fast and easy!

- Fast, Single Pass Compiler
- Powerful Text Editor
- Fast Single Pass Linker
- Source Level Debugger
- 170+ Libraries in source form including ARP, ARexx, IFF and C

**Fax , Call, or Write for *FREE* Demo Disk!**

***M2S, Inc.***
Box 550279 • Dallas, Texas 75355
Phone (214) 340-5256 • Fax (214) 341-9104

Demo also available on BIX (see M2Demo.zoo in M2S/listings section) and CompuServe (see M2Demo.zoo in Amigavendor forum, section 12).

Amiga is the registered trademark of Commodore Business Machines

## Elan Presents...

**Puts Amiga imagery at your fingertips**: Elan Design Inc., brings *Elan Performer*, graphics presentation software for the Amiga. Elan Performer allows you to display all your Amiga grahics quickly, at the touch of a key.

With the aid of a simple, interactive control screen, images or animations are assigned to keys on the keyboard. Then press any key and its associated image is instantly displayed.

Displays can also be fully automated to follow precise timing and sequence settings. As well, the mouse can be used as a 'remote control', allowing movement backward and forward through the imagery.

Elan Performer supports IFF, HAM, RIFF, ANIM and RGB formats as well as all Amiga resolutions. The list price is $59.00 US.

**Contact**: Elan Design, P.O. Box 31725, San Francisco, CA 94131, USA.

## Twin-X from ASDG

**ASDG links Amiga to the world**: ASDG's *Twin-X* plugs into the expansion slots inside the Amiga 2000 or 2500 and connects to different industry standard modules. These in turn link the Twin-X and the Amiga to a list of consumer, industrial, medical and scientific equipment..

ASDG currently offers three add-on modules: the *SBX-Serial/2* and the *RS-232 Serial Modules* and an *SBX-GPIB* module that conforms to the IEEE-488 industrial interface standard.

Since each Twin-X module is designed to an industry standard specification, Twin-X can now be used with over 200 different modules from over two dozen manufacturers.

The Twin-X Input/Output Board lists at $329.95 US. Both the SBX-GPIB module and SBX-Serial/2 retail for $199.95 each.

**Contact**: ASDG Incorporated, 925 Stewart St., Madison, WI 53731, USA

*Elan Performer allows you to display all your Amiga grahics quickly, at the touch of a key...*

*Many of the exotic input/output functions needed in today's industrial or research settings can now be plugged onto a Twin-X board and used with the Amiga...*

## TASS: New ARexx Development Tool

*In order to encourage the widest possible use of TASS, Mindware has released a subset of TASS into the public domain, and will soon make it available through major disk magazine publications and on-line networks...*

**New ARexx application development tool**: Mindware International has released a subset of the new *Thut Application Support System* (TASS) into the public domain.

TASS is a extensive library of software modules containing a wide range of frequently used programming elements and interface design tools for the applications developer who is using ARexx to integrate commercial software packages.

Created as a 'hub', or centre of activity for ARexx integration, TASS lets developers build an interface for the end user that implements ARexx macros for use with the growing number of commercial programs that now support ARexx.

Thanks to TASS, the end user will no longer be required to learn ARexx programming in order to take advantage of it. All the end user sees is a mouse driven TASS screen that conceals the ARexx macros running behind the interface. Yet the full power of ARexx remains available.

The use of TASS as a development toool will not only allow ARexx programmers to shorten development time, it will also ensure that all ARexx-based applications will share a long list of standard features, together with a unified, familiar look and feel, involving gadgets and requester windows.

In order to encourage the widest possible use of TASS, Mindware has released a subset of TASS into the public domain, and will soon make it available through major disk magazine publications and on-line networks.

The same subset of the TASS library is also available for $10.00 Cdn. by writing directly to the company.

**Contact**: Mindware International, 110 Dunlop St. W., Box 22158, Barrie, ON, CAN, L4M 5R3.

## NewTek Upgrades Digi-Paint

**Digi-Paint turn three**: NewTek has announced the release of *Digi-Paint 3*, a completely new and rewritten version of their paint program for the Amiga.

Some of Digi-Paint's new features include full Amiga font support with anti-aliasing of text, an extended set of drawing tools with resizable built-in brushes, and user-controllable 3D texture mapping.

This last feature allows the user to control the amount of brush warping, the 'warping point' of the brush and the brush transparency. New display features include support for auto-scrolling bitmaps up to 1024x1024 pixels, user-definable overscan, and changing the screen resolutions from the program.

Digi-Paint 3's dithering routines make use of 30 bytes per pixel to calculate colours, giving the program a universe of over 1 billion virtual colours from which to create the best dithering patterns. Other features include new drawing modes, an improved magnify mode, a larger palette area, unlimited brush resizing with anti-aliasing, brush rotation and range painting.

For current Digi-Paint owners, NewTek is offering an upgrade to Digi-Paint 3. Owners should send the last page of their manual and $29.95 US (plus $6.00 US shipping and handling) to: Digi-Paint 3 Upgrade, 115 W. Crane St., Topeka, KS 66603.

Owners of *DeluxePaint II*, *Photon Paint*, or *DeluxePhotoLab* can send in any page from their manual plus $49.95 US (plus $6.00 US shipping and handling) to NewTek to receive their complete copy of Digi-Paint 3.

Digi-Paint has a suggested retail value of $99.95 US.

**Contact**: NewTek, Inc., 115 West Crane St., Topeka, KS 66603, USA.

*Digi-Paint 3's dithering routines make use of 30 bytes per pixel to calculate colours, giving the program a universe of over 1 billion virtual colours from which to create the best dithering...*

## Even more for ProPage

*Coming from Gold Disk in September will be V1.3 of* Professional Page. *This upgrade will include* Compugraphic *outline font technolgy...*

**Gold Disk supplements ProPage**: Templates and structured clipart have been added to the Professional series with *Professional Page Templates and Design Guide* and *Structured ClipArt*. Both of these products have been designed for use with Professional Page V1.2; the second can also be used with ProDraw.

Professional Page Templates and Design Guide contains over fifty professionally-designed templates, as well as a 48-page design manual full of design and layout tips.

The Structured ClipArt package contains dozens of structured graphics covering a variety of topics: sports, holidays, maps, food, basic symbols, etc. These structured graphics will output to the highest resolution of any printer device regardless of scaling.

Included with the package is a utility which allows these graphics to be converted to bitmaps to be used in any Amiga program supporting the IFF graphic standard. Both products list at a suggested $59.95 Cdn.

Also, coming from Gold Disk in September we can expect V1.3 of *Professional Page*. This upgrade will include the long awaited *Compugraphic* outline font technolgy, as well as offering support for dot-matrix, inkjet, and Laserjet compatible printers to their highest resolution.

Another goodie expected for Version 1.3 will be direct on-line support for Gold Disk's *Transcript* word processor. Desktop publishers will be able to quickly and easily transfer text from their on-screen layouts to Transcript and then back again into Professional Page.

Professional Page 1.3 will list at $395.00 US, $495.00 CDN.

**Contact**: Gold Disk, P.O. Box 789, Streetsville, Mississauga, ON, CAN, L5M 2C2.

## Networking Solutions

**An apple for the teacher**: Creative Microsystems Inc. now offers an affordable *MultiPort Board* and *CMI-net Utilities Software* for the Amiga. Installation of the CMI-net allows networking of Amigas to a Macintosh network or to an Amiga network.

The heart of the CMI-net consists of the MultiPort Board which is easily installed into an expansion slot of the Amiga 2000. The optional CMI-net Utilities Software provides all features necessary to complete the network. The MultiPort Board and the software are purchased separately.

The MultiPort Board is an expansion board which contains an RS-232 Serial Port and an RS-422 8-pin DIN Apple Macintosh Serial Port. It also houses a DB-25 parallel printer port.

Two optional features are offered on the board at an added cost. These include a SCSI hard disk interface and a 2-Mb Supra RAM expansion board available for the Amiga 500 and 1000. The addition of the MultiPort itself will free up the built-in ports on the Amiga

CMI-net Utilities Software is a complete networking, connectivity solution. All that is needed to complete a fully functional network is the MultiPort Board.

Hook the board up to AppleTalk connectors and run the CMI-net Utilities Software. CMI-net Utilities Software includes an AppleTalk protocol utility which allows the Amiga to be integrated into a Macintosh network.

In addition to its networking features, CMI-net also contains a Laser Printer driver which will allow the Amiga to use an AppleTalk Protocol Laser Printer. No prices currently available from Creative Microsystems.

**Contact**: Creative Microsystems Inc., 19552 SW 90th Court, Tualatin, OR 97062, USA.

*The MultiPort Board is an expansion board which contains an RS-232 Serial Port and an RS-422 8-pin DIN Apple Macintosh Serial Port...*


VISA

# Programmer's Shareware

MasterCard

### Programming Languages/Toolkits

- ☐ **Assemblers and Structured Programming Tools (ALA101)**
  **2 Macro Assemblers and 2 Structure Preprocessors!**
- ☐ **Debuggers/System Monitors (ALA102)**
  **Great for fixing a bug in your program or just exploring!**
- ☐ **Assembly Language Examples, Toolkits, Utilities (ALA103)**
  **Examples and libraries for Assembly, with source!**
- ☐ **C Language Utilities (ALC101)**
  **Make utility, execution profiler, C tutorial and more!**
- ☐ **Optimizing C Language Compiler (ALC102)**
  **Full featured C compiler with SOURCE CODE!**
- ☐ **C Language Examples, Toolkits and Utilities (ALC103)**
  **Examples and libraries for C, with source!**
- ☐ **Artificial Intelligence Languages (ALM101)**
  **Object-oriented Lisp and a Prolog interpreter with source!**
- ☐ **Misc. Programming Languages (ALM102)**
  **LOGO graphics, ICON and a PostScript interpreter!**
- ☐ **Modula-2 Utilities Disk (ALM103)**
  **Asm & C to Modula-2 conversion, and a Modula-2 library!**

### Programmers Editors/Word Processing

- ☐ **WordWright Word Processor (AWP101)**
  **Word processor with outliner, mailmerge and Spell check!**
- ☐ **Programming and Text Editors (AWP102)**
  **3 Great editors! Multiple windows, LARGE files, etc.**

### Utilities

- ☐ **Archive & Compression Utilities (AUT101)**
  **Put twice as much information on your disks!**
- ☐ **Anti-Virus & Vaccine Utilities (AUT102)**
  **Protect your Amiga from Computer Viruses!**
- ☐ **Printer Utilites (AUT103)**
  **Eight utilities and Printer Drivers for over 40 printers!**
- ☐ **Great Utilities Disk #1 (AUT104)**
  **Disk recovery, disk optimizer, task handling, and more!**
- ☐ **Great Utilities Disk #2 (AUT105)**
  **Programmers WorkBench, Calculator, Shell and more!**
- ☐ **Diagnostics/Error Detection and Prevention (AUT106)**
  **Hardware test, Disk test, Guru error prevention!**
- ☐ **ARP - AmigaDOS Replacement Project (AUT107)**
  **Add power, speed and more commands to AmigaDOS!**

### Recreational/Personal/Business

- ☐ **RIM DataBase Management System (ABP106/ABP107)**
  **Relational Information Management System. With source!**
- ☐ **AnalytiCalc Spreadsheet (ABP104/ABP105)**
  **Large and Powerfull with many extra features!**
- ☐ **Great Games Volume I (AGA102)**
  **StoneHenge, Missile Command, MazeMan and more!**
- ☐ **Great Games Volume II (AGA103)**
  **Gravity Wars, BattleShip, BlackJack, Clue and more!**

**TOLL FREE ORDERING: 800/874/9375**
**Canada, MI, & Info: 517/628/2943**
**Or Send Check or Money Order.**
**Visa and MasterCard accepted.**

**Computer Solutions**
**P.O. Box 354**
**Mason, MI 48854**

**1-9 disks -$5.50 per disk**
**10 or more - $5.25 per disk**
**Small Order (less $20.00) add $3.00**
**MI residents add 4% sales tax**

**MAIL ORDERS BE SURE TO INCLUDE COMPLETE ADDRESS, DAYTIME PHONE AND CHECK OR CREDIT CARD NUMBER!**

Fill out this card (please print) or attach your peel-off mailing label, and circle the reader service number for the ads that you'd like more information about (see Advertiser Index below).

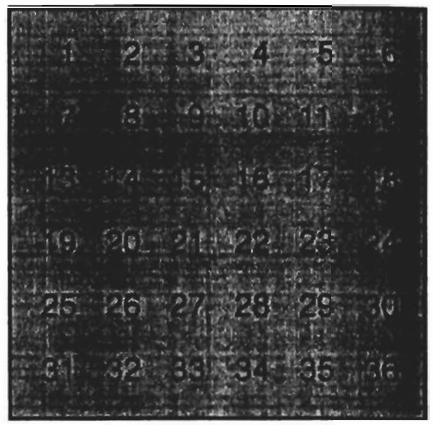
*Name* _____

*Company* _____

*Address* _____

_____

*Postal/Zip Code* _____



TA V3I1

| Advertiser | Page | Reader Service # |
|---|---|---|
| ASDG Inc. | 23 | 1 |
| Black Belt Systems | 39 | 2 |
| CompuServe | 31 | 3 |
| Computer Solutions | 80 | 4 |
| CSA (Computer Systems Associates) | 10 | 5 |
| Consultron | 15 | 6 |
| Delphi Noetic Systems Inc. | 66 | 7 |
| Delta Research | 38 | 8 |
| Electronics 2001 | 61 | 9 |
| Free Spirit Software Inc. | 19 | 10 |
| Imtronics Inc. | 5 | 11 |
| Imtronics Inc. | 39 | 12 |
| Innerprise | Inside Back Cover | 13 |
| Inovatronics Inc. | 41 | 14 |
| Intercomputing Inc. | 7 | 15 |
| Lake Forest Logic Inc. | 19 | 16 |
| Lattice Inc. | Inside Front Cover | 17 |
| M2S Inc. | 77 | 18 |
| Microway | Outside Back Cover | 19 |
| Newtek Inc. | 15 | 20 |
| The Grapevine Group | 69 | 21 |
| The Puzzle Factory | 23 | 22 |
| The Software Shop | 49 | 23 |
| Vidtech International Inc. | 1 | 24 |
| Virtual Reality Laboratories | 73 | 25 |

# for the AMIGA Transactor

**The Magazine for Amiga Programmers**

Please print your name and address below (or transfer your mailing label)

Check here for change of address ☐

*Name:* _____

*Address:* _____

*City:* _____ *Province/State:* _____

*Zip/Postal:* _____ *Phone:* _____ *CompuServe ID:* _____

**NOTE: Prepayment required. Purchase orders will be accepted ONLY if accompanied by payment.**

Subscribers only: please enter subscription number from mailing label ☐

## Subscription Order Section

**PLEASE NOTE THAT TRANSACTOR FOR THE AMIGA IS NOW PUBLISHED MONTHLY**

| | Canada ($Cdn.) | | | U.S.A. ($US) | | | Foreign ($US) | | | Air Mail ($US, overseas only) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mag | Disk* | Both* | Mag | Disk | Both | Mag | Disk | Both | Mag | Disk | Both |
| 12 issues | ☐35.00 | ☐123.00 | ☐158.00 | ☐27.50 | ☐101.00 | ☐128.50 | ☐38.50 | ☐119.50 | ☐157.50 | ☐ 73.00 | ☐ 145.00 | ☐ 218.00 |
| 24 issues | ☐65.00 | ☐220.00 | ☐270.00 | ☐49.00 | ☐160.00 | ☐190.00 | ☐69.00 | ☐220.00 | ☐290.00 | ☐ 135.00 | ☐ 250.00 | ☐ 390.00 |

* Ontario residents please add 8% of disk subscription price for provincial sales tax. No PST on magazine subscriptions.

## Combo Subscription Bonus!

**Subscribe to BOTH the magazine and the disk and receive a FREE extra issue on your subscription!**

## Product Order Section

Ontario residents please add 8% PST on all items

| $Cdn. | $U.S. | |
|---|---|---|
| ☐ 14.95 | 12.95 | Transactor Classic Amiga Disk #1 |
| ☐ 14.95 | 12.95 | Transactor Classic Amiga Disk #2 |

**Fish Disks $6.95 US, $7.95 Cdn. each**

Note: Disk numbers not yet released may be ordered in advance - simply add numbers to end of list. Disks 57, 80 and 88 are not available.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ☐ 1 | ☐ 2 | ☐ 3 | ☐ 4 | ☐ 5 | ☐ 6 | ☐ 7 | ☐ 8 | ☐ 9 | ☐ 10 | ☐ 11 | ☐ 12 | ☐ 13 | ☐ 14 |
| ☐ 15 | ☐ 16 | ☐ 17 | ☐ 18 | ☐ 19 | ☐ 20 | ☐ 21 | ☐ 22 | ☐ 23 | ☐ 24 | ☐ 25 | ☐ 26 | ☐ 27 | ☐ 28 |
| ☐ 29 | ☐ 30 | ☐ 31 | ☐ 32 | ☐ 33 | ☐ 34 | ☐ 35 | ☐ 36 | ☐ 37 | ☐ 38 | ☐ 39 | ☐ 40 | ☐ 41 | ☐ 42 |
| ☐ 43 | ☐ 44 | ☐ 45 | ☐ 46 | ☐ 47 | ☐ 48 | ☐ 49 | ☐ 50 | ☐ 51 | ☐ 52 | ☐ 53 | ☐ 54 | ☐ 55 | ☐ 56 |
| ☐ 58 | ☐ 59 | ☐ 60 | ☐ 61 | ☐ 62 | ☐ 63 | ☐ 64 | ☐ 65 | ☐ 66 | ☐ 67 | ☐ 68 | ☐ 69 | ☐ 70 | ☐ 71 |
| ☐ 72 | ☐ 73 | ☐ 74 | ☐ 75 | ☐ 76 | ☐ 77 | ☐ 78 | ☐ 79 | ☐ 81 | ☐ 82 | ☐ 83 | ☐ 84 | ☐ 85 | ☐ 86 |
| ☐ 87 | ☐ 89 | ☐ 90 | ☐ 91 | ☐ 92 | ☐ 93 | ☐ 94 | ☐ 95 | ☐ 96 | ☐ 97 | ☐ 98 | ☐ 99 | ☐ 100 | ☐ 101 |
| ☐ 102 | ☐ 103 | ☐ 104 | ☐ 105 | ☐ 106 | ☐ 107 | ☐ 108 | ☐ 109 | ☐ 110 | ☐ 111 | ☐ 112 | ☐ 113 | ☐ 114 | ☐ 115 |
| ☐ 116 | ☐ 117 | ☐ 118 | ☐ 119 | ☐ 120 | ☐ 121 | ☐ 122 | ☐ 123 | ☐ 124 | ☐ 125 | ☐ 126 | ☐ 127 | ☐ 128 | ☐ 129 |
| ☐ 130 | ☐ 131 | ☐ 133 | ☐ 134 | ☐ 135 | ☐ 136 | ☐ 137 | ☐ 138 | ☐ 139 | ☐ 140 | ☐ 141 | ☐ 142 | ☐ 143 | ☐ 144 |
| ☐ 145 | ☐ 146 | ☐ 147 | ☐ 148 | ☐ 149 | ☐ 150 | ☐ 151 | ☐ 152 | ☐ 153 | ☐ 154 | ☐ 155 | ☐ 156 | ☐ 157 | ☐ 158 |
| ☐ 159 | ☐ 160 | ☐ 161 | ☐ 162 | ☐ 163 | ☐ 164 | ☐ 165 | ☐ 166 | ☐ 167 | ☐ 168 | ☐ 169 | ☐ 170 | ☐ 171 | ☐ 172 |

**The Transactor for the *AMIGA* Disk $12.95 US, $14.95 Cdn. each**

(these disks correspond to issues of **Transactor for the *AMIGA***)

☐ Vol. 1, Iss. 1 ☐ Vol. 1, Iss. 2 ☐ Vol. 1, Iss. 3 ☐ Vol. 1, Iss. 4 ☐ Vol. 1, Iss. 5 ☐ Vol. 1, Iss. 6

☐ Vol. 2, Iss. 1 ☐ Vol. 2, Iss. 2 ☐ Vol. 2, Iss. 3 ☐ Vol. 2, Iss. 4 ☐ Vol. 2, Iss. 5 ☐ Vol. 2, Iss. 6

**Add flat $2.00 shipping & handling for any number of items ordered from above box.**

**Back Issues: $4.50 U.S., $4.50 Cdn. (filled box=SOLD OUT)**

☐ Vol. 1, Iss. 1 ■ Vol. 1, Iss. 2 ☐ Vol. 1, Iss. 3 ☐ Vol. 1, Iss. 4 ☐ Vol. 1, Iss. 5 ☐ Vol. 1, Iss. 6

☐ Vol. 2, Iss. 1 ☐ Vol. 2, Iss. 2 ☐ Vol. 2, Iss. 3 ☐ Vol. 2, Iss. 4 ☐ Vol. 2, Iss. 5 ☐ Vol. 2, Iss. 6

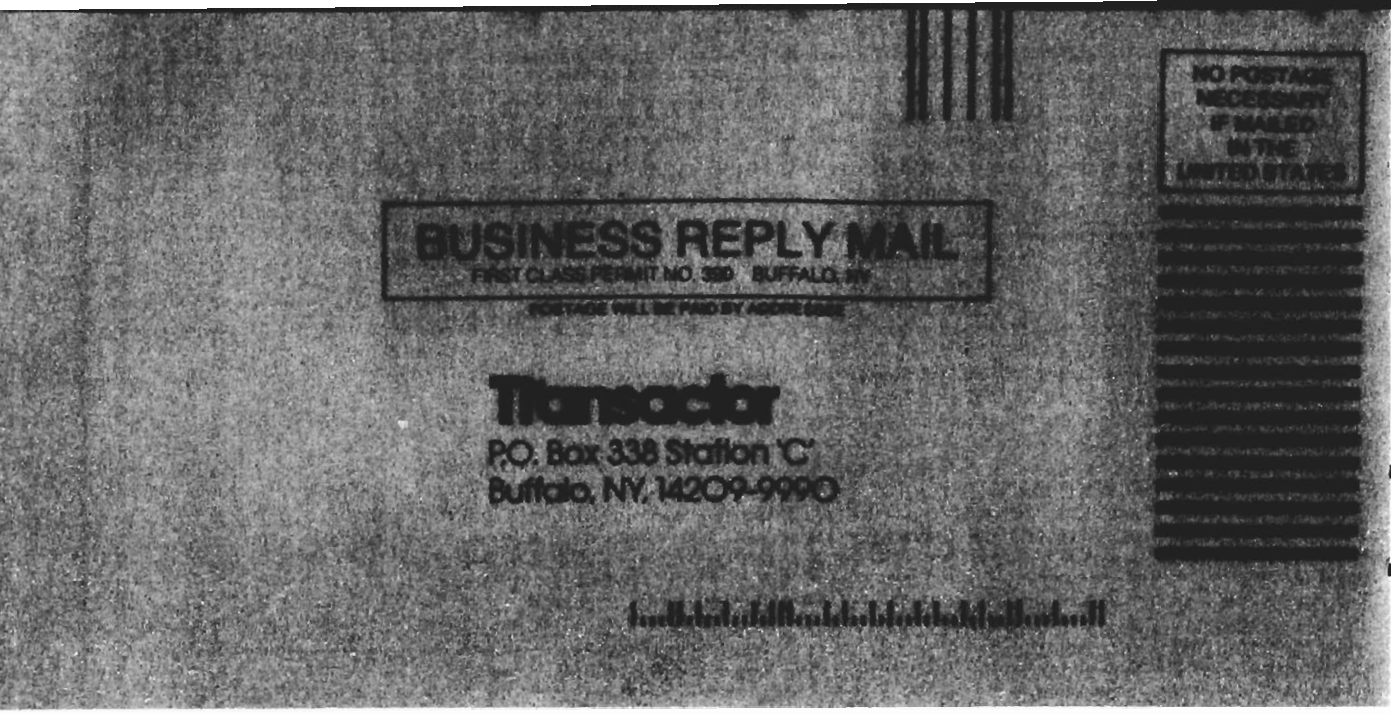
**Ontario residents please add 8% provincial sales tax for items ordered from this box.**

☐ Visa ☐ MasterCard ☐ Cheque/MO enclosed Cheque #_____ Dated ____/____/____ Amount _____

Acct. # _____ Expires ____/____ Signature _____

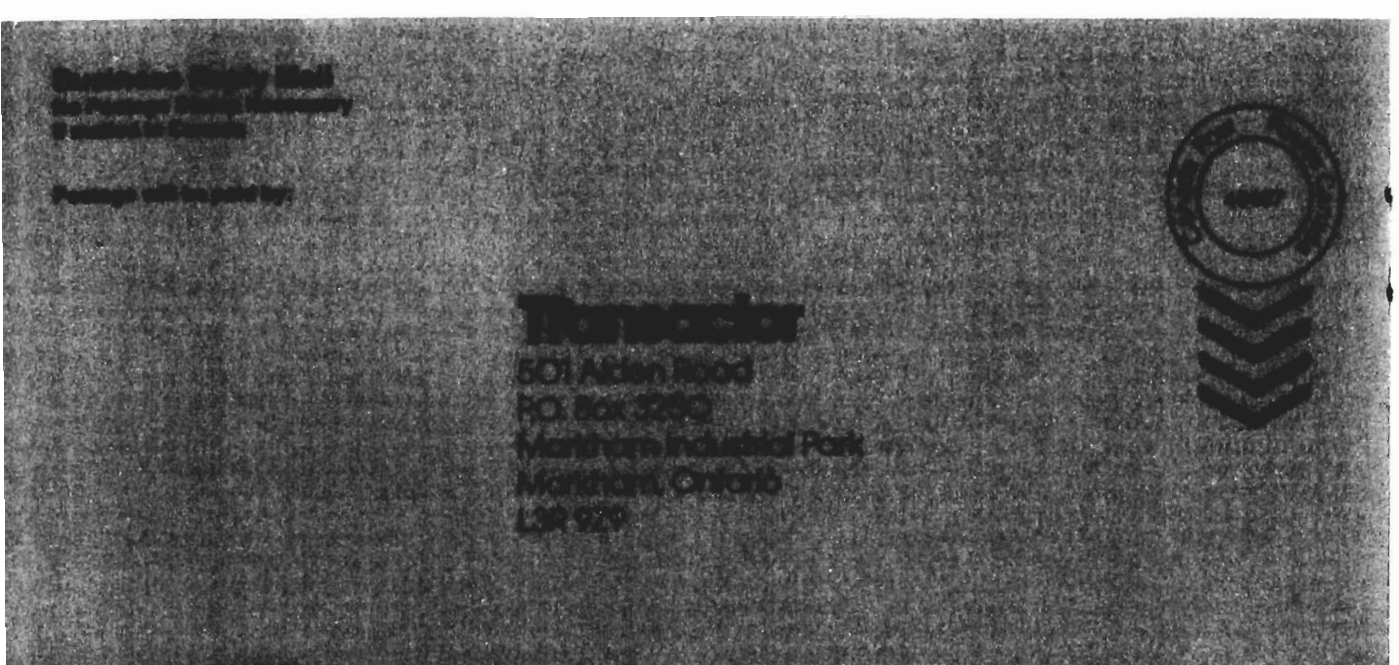
**Please allow 6-8 weeks for delivery in the U.S., 4-6 weeks in Canada.**

FOLD ALONG SCORES WITH CORRECT REPLY CARD ON OUTSIDE. USE TAPE, PLEASE - NO STAPLES OR GLUE.




place
stamp
here

**Transactor**
10-85 West Wilmot Street
Richmond Hill, Ontario, Canada
L4B 1K7




**Advertising in Transactor for the Amiga**

Interested in advertising in *TransAmi?* Call Mike Grantham at 416/764-5274

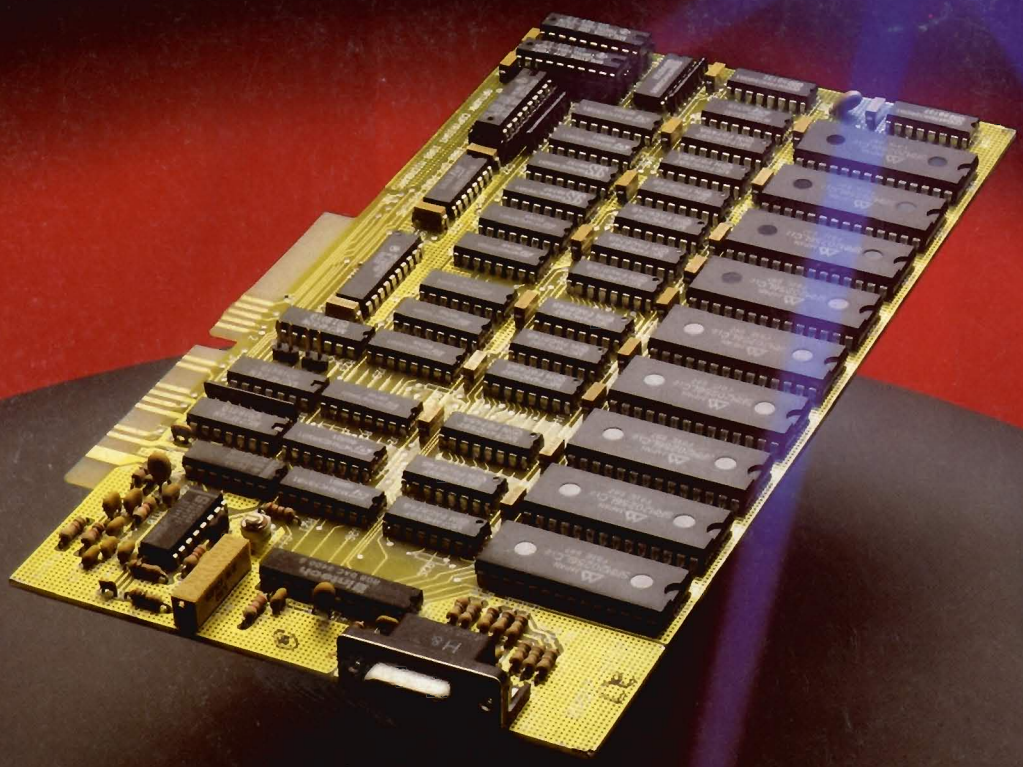
Use our convenient...

**TOLL-FREE ORDER LINE**
**1-800-248-2719 Extension 911**

Here's a convenient way to order magazine or disk subscriptions and other products. Nothing to fill out or drop in the mail! Just call the toll-free number above and have your credit card number ready. Available in the United States only.

NOTE: The toll-free number can be used for ordering only; for subscription queries, etc., call or write to our office directly.

## AGA 2000
## The flickerFixer™



# UNLOCK THE GRAPHICS POWER OF YOUR AMIGA 2000!

***flickerFixer*** is an advanced graphics adaptor that eliminates your Amiga 2000's interlace flicker and visible scan lines. The result: superior quality color or monochrome graphics and text — for such demanding applications as CAD/CAM, Desktop Presentation, Graphic Design, Animation, 3D Modeling, Video, and Word Processing.

***flickerFixer*** upgrades the Amiga 2000 with a flicker free 4096 color palette, has an overscan mode that features a screen size of 704 x 470 pixels and drives most of the popular PC Multiscan and VGA monitors, including the NEC Multisync and Mitsubishi XC1429C.

Accolades include: ***Best of 1988 Award,*** Commodore Magazine (12/88); ***#1 Reader's Choice Award***, AMIGAWORLD (12/88); ***"The display is fantastic . . . It is the best display we have ever seen on any computer system."*** Amiga GURU (5/88)

***flickerFixer*** fits into the Amiga video slot, is fully compatible with all software, and does not modify the standard Amiga video signals. For more information or to order, call MicroWay Sales at (508) 746-7341 or your Amiga Dealer. Priced at $595, ***flickerFixer*** is made in the USA and is FCC Class B approved.

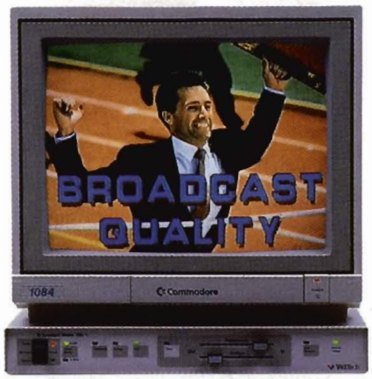
***MicroWay . . . Respected throughout the industry for high quality engineering, service and technical support.***

**Micro Way**

*World Leader in PC Numerics*

*P.O. Box 79. Kingston, MA 02364 USA (508) 746-7341*
*32 High St., Kingston-Upon-Thames, UK, 01-541-5466*
*USA FAX 617-934-2414 Australia 02-439-8400*

flickerFixer and MicroWay are trademarks of MicroWay, Inc. Amiga is a registered trademark of Commodore. Multisync is a registered trademark of NEC.



# Coming To A Screen Near You.

If you're ready to become your own studio producer, there's only one desktop video system that will bring NTSC and S-VHS to your Amiga's screen. The Scanlock system by VidTech. Other systems give you a genlock that's "either-or." They either support NTSC or S-VHS. But not both.

**Two For One**

With Scanlock, you get two genlocks for the price of one. So you can migrate from NTSC to the newer S-VHS format without investing in another system. We also offer a Scanlock model that is compatible with the International PAL TV format, including its new S-VHS version.

Scanlock's dual-genlock system gives you broadcast-quality video at consumer prices. And our system offers all the bells and whistles, including front-panel fingertip controls, multiple monitor connections for viewing and editing, and independent fade controls for both reference and Amiga videos. If your Amiga is supporting a load of peripherals, we provide a connection that lets you use an optional external power supply.

Scanlock is a breeze to use. A "glitch-free" vertical interval switch lets you insert, reverse and remove reference video and Amiga graphics at the touch of a finger. And you can switch between NTSC, S-VHS and analog RGB formats without moving cable connectors. When you're done editing, you can resume normal Amiga operation with the flip of a switch.

**Don't Get Locked Out**

So if you're ready to become your own producer, choose the desktop video system that doesn't lock you into one genlock — the Scanlock system by VidTech. Coming to a screen near you.

Call or write us today: VidTech International, Inc., 2822 NW 79th Ave., Miami, Florida 33122. Telephone 800-727-2261 or 305-477-2228. Fax 305-591-1651. Prices:NTSC-$995, PAL-$1,095. Distributed by Southern Technologies, Dallas, Texas.

VidTech